eoe·Android开发者门户鼎力支持

134个典型实例、2个大型案例，实战Android开发



高彩丽 许黎民 袁 海 等编著

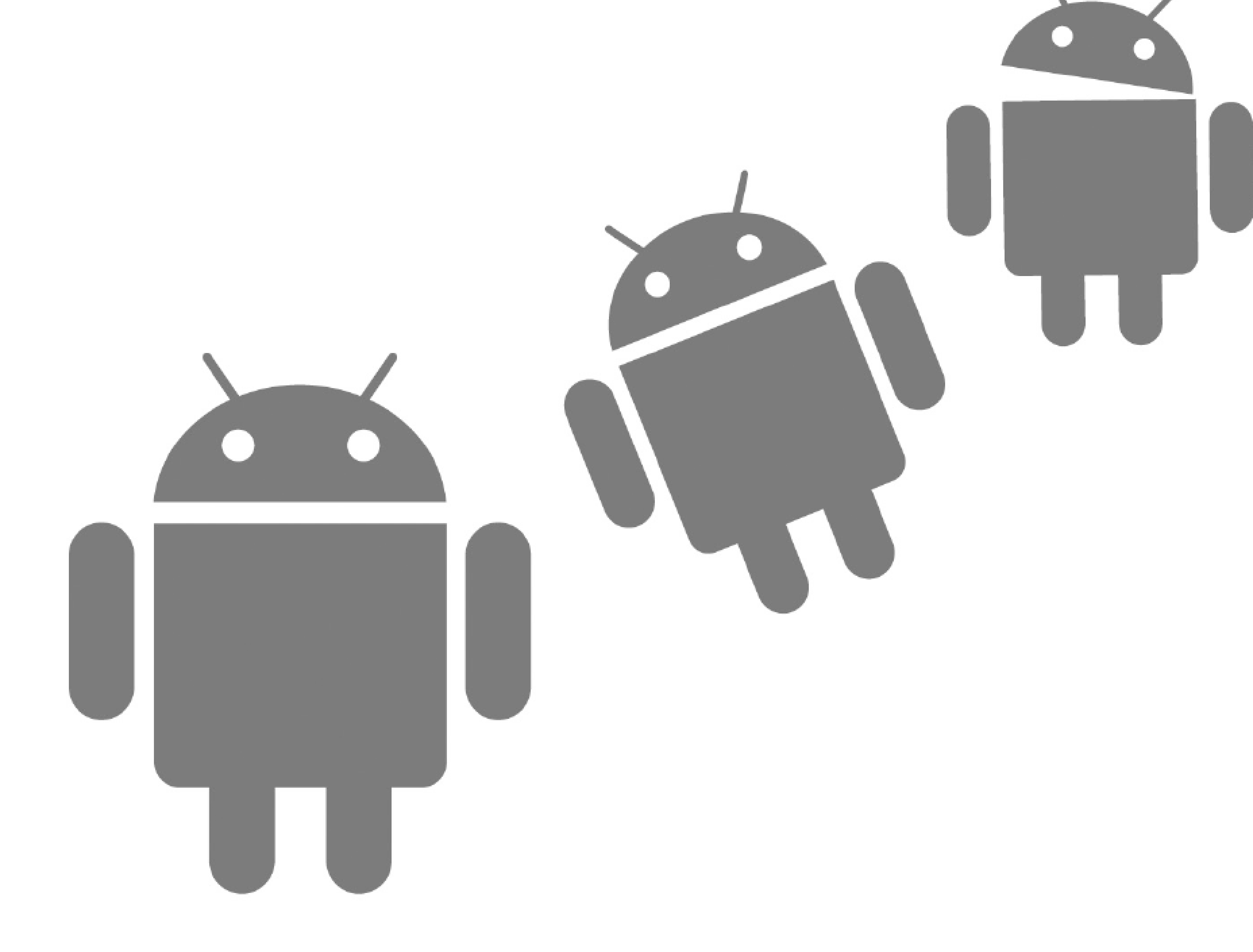
Android 应用开发范例精解

清华大学出版社

eoe·Android开发者门户鼎力支持

134个典型实例、2个大型案例，实战Android开发





高彩丽 许黎民 袁 海 等编著

Android 应用开发范例精解

清华大学出版社
北 京

内 容 简 介

本书通过通俗易懂的开发实例及项目案例，详细介绍了 Android 应用开发的知识体系及实用开发技术。

本书共 14 章，分为 3 篇。第 1 篇为基础篇，涵盖 Android 背景及开发环境和 Android 常用工程组件。第 2 篇为应用开发篇，通过实例介绍了 Android UI 布局、Android 人机界面、手机硬件设备的使用、Android 本地存储系统、Android 中的数据库、多线程设计、Android 传感器、Android 游戏开发基础、Android 与 Internet，以及 Google 地图服务等内容。第 3 篇为项目案例实战篇，详细介绍了 Android 地图定位搜索应用及乐乐网上购物商城两个案例的实现过程。

本书的最大特色是实用性强。书中的每一个知识点都通过通俗易懂、使用频率比较高的实例进行讲解，还提供了项目实战案例，可以使读者能够快速地掌握 Android 应用开发。本书适合有一定 Java 基础的移动开发人员阅读，也适合作为相关院校和社会培训机构的教材。

图书在版编目（CIP）数据

Android 应用开发范例精解 / 高彩丽等编著. —北京：清华大学出版社，2012.1
ISBN 978-7-302-27600-5

Ⅰ. ①A…　Ⅱ. ①高…　Ⅲ. ①移动终端－应用程序－程序设计　Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字（2011）第 265895 号

责任编辑： 夏兆彦
责任校对： 徐俊伟
责任印制：
出版发行： 清华大学出版社　　**地　　址：** 北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 185×260　　**印　　张：** 28.25　　**字　　数：** 706 千字
（附光盘 1 张）
版　　次： 2012 年 1 月第 1 版　　**印　　次：** 2012 年 1 月第 1 次印刷
印　　数： 1～ 000
定　　价： 　　元

产品编号：043696-01

前　言

在资讯传播速度越来越快的今天，人们希望可以随时随地地获取信息。随着智能手机技术的日益成熟，手机自然成为人们获取信息的首选通信工具。所以我们可以看到，越来越多的基于手机的应用程序被开发出来。很显然，手机应用开发已成为日益重要的领域。

目前，在智能手机操作系统领域，诺基亚公司的塞班系统已经开始落伍，逐渐被淘汰，而由 Google 公司开发的 Android 系统则成了当前这一领域最为热门的角色。Android 系统是一个 Linux 平台的开源手机操作系统，于 2007 年 11 月 5 日公布，目前已获得了 HTC、三星、索爱、摩托罗拉等大批手机厂商的支持。

为了让开发人员可以迅速地掌握 Android 应用程序开发，我们编写了本书。本书侧重于实战开发，从 Android 开发环境的搭建开始讲解，涵盖了 Android 程序的界面布局、控件使用、手机资源调用、网络访问和地图位置热门技术。书中的知识点都是通过一个个实际的应用实例来讲解，读者可以轻松地掌握实现需求效果的技术细节。本书的最后一篇给出了两个大型项目案例的实现过程，可以提高读者的实战开发水平。

本书有何特色

1．实例带动技术讲解，实用性强，且容易上手

本书在内容选择和安排上，都从实际应用出发，每章都以实际案例带动技术讲解，并配以源代码和效果图，能够让读者快速入门、快速上手。在案例的选择上，注重由浅入深，突出重点，让当前技术要点一目了然，明确直观。

2．实例丰富、典型，容易掌握

大多数情况下，开发者在实际开发中更关注 “如何实现效果”这一需求。因此本书提供了丰富、典型的实例，以满足读者的这一需求。这些实例涉及 Android 开发的方方面面，读者可以把本书当作一本工具书，轻易地找到自己所关心的技术细节的实现。

3．提供大型案例，注重项目实战

本书最后提供了两个大型项目案例的实现过程，并详解案例的关键代码。通过这两个案例的学习，读者可以举一反三，大幅提高实战开发水平。

4．通俗易懂，步骤详细

本书中每个实例和项目案例的实现步骤都以通俗易懂的语言阐述，并穿插必要的技巧

讲解，每个例子都提供了相应的效果图，像有位老师在时刻指导读者学习。读者只需要按照书中的步骤，便可实现所有的实例效果，并能独立完成相应的开发。

本书内容导读

本书共 14 章内容，分为 3 篇。各章内容介绍如下：

第 1 章介绍 Android 的背景与开发环境，包括 SDK 的下载、ADT 与 MyEclipse 的整合、模拟器的创建等。

第 2 章对 Android 的四大工程组件进行了介绍。其中对 Activity 的介绍是重点。

第 3 章介绍 Android 程序的 UI 布局，包括使用 XML 创建的各种布局和由 Java 代码创建的自定义布局。

第 4 章介绍 Android 的各种控件，包括控件的各种使用方法、参数设定及特殊效果等。

第 5 章介绍 Android 调用手机自身的资源，包括调用媒体播放器、电话、短信、蓝牙及摄像头等。

第 6 章介绍 Android 使用手机的本地存储功能，包括对 SD 卡上的文件进行读写操作。

第 7 章介绍 Android 系统中内置的数据库 SQLite 的使用。

第 8 章介绍如何在 Android 应用程序中进行多线程开发。

第 9 章介绍如何在 Android 系统中调用手机自带的传感器进行应用程序的开发。重点介绍加速度传感器和方向传感器。

第 10 章介绍 Android 系统的游戏开发框架，包括使用 View、SurfaceView 框架绘图及动画操作等。

第 11 章介绍 Android 系统对网络的访问操作，包括使用内置的浏览器、发送 POST/GET 请求、解析 XML/JSON 数据及上传下载文件等。

第 12 章介绍在 Android 应用程序中使用 Google 地图服务，包括地图定位、地点标注和地理查询等操作。

第 13、14 章是两个大型的应用案例。一个是读取本地文件及数据库信息的地理线路描述程序；另一个是获取网站信息的网络购物手机客户端。

本书读者对象

本书内容全面，实例精彩，指导性强，涵盖 Android 开发的所有重点内容。本书适合以下读者阅读：

- ❑ Android 初学人员；
- ❑ 有一定 Java 基础的移动开发人员；
- ❑ 由 Java 开发转 Android 开发的人员；
- ❑ 作为案头工具书的移动开发人员。

本书作者

本书由高彩丽、许黎民、袁海主笔编写。其他参与编写的人员有毕梦飞、蔡成立、陈涛、陈晓莉、陈燕、崔栋栋、冯国良、高岱明、黄成、黄会、纪奎秀、江莹、靳华、李凌、李胜君、李雅娟、刘大林、刘惠萍、刘水珍、马月桂、闵智和、秦兰、汪文君、文龙。

编著者

目　录

第 1 篇　Android 开发基础

第 2 篇　Android 应用开发实例

第 3 篇　Android 项目案例实战

第 1 篇　Android 开发基础

第 1 章　Android 背景及开发环境介绍

Android 在英文中本义是指“机器人”，它是 Google 公司于 2007 年 11 月宣布的基于 Linux 平台的开源手机操作系统。该系统由底层的 Linux 操作系统、中间件和核心应用程序组成。

Android 是基于 Java 并运行在 Linux 内核上的操作系统，Android 应用程序使用 Java 语言编写，也支持其他一些语言，如 C、Perl 等语言。

1.1　Android 背景介绍

为了更好地学习 Android，有必要了解其历史背景。Android 早期是由原名为 Android 的公司开发，后来 Google（谷歌）在 2005 年收购 Android，并继续对其进行开发运营。Google 在 2007 年 11 月 5 日发布了 Android 1.0 手机操作系统，并且组建了一个全球性的联盟组织“开放手机联盟”，其英文名称为 Open Handset Alliance。开放手机联盟主要包括手机制造商、手机芯片厂商和移动运营商等几类。

2007 年 11 月 12 日 Google 发布了能在 Windows、Mac OS X、Linux 等多平台上使用的 Android 开发工具 SDK 与其相关文件，并且可以免费下载。随后，Google 再次发布作业系统核心与部分驱动程序的源代码。

2008 年 9 月 24 日，T-Mobile 首度公布第一台 Android 手机（G1）的细节，Google 也发布了 Android SDK 1.0 rc1。Android SDK 1.0 rc1 代表了开发者可以放心、安全地使用 API，不必担心 API 有太大的变动。

2008 年 10 月 21 日，Open Handset Alliance 公开了全部 Android 的源代码，至此，一个完全开放的手机平台向开发者敞开了大门。

1.2　Android 开发环境概述

Android SDK 提供了一系列工具，包括模拟硬件设备的模拟器（Emulator）、Android 资源打包工具 AAPT（Android Asset Packaging Tool）、Dalvik 调试监视服务 DDMS（Dalvik Debug Monitor Service）、Android 调试桥 adb（Android Debug Bridge）和将.class 字节码文件转换为.dex 文件的 DX 工具等。

使用上述这些工具，可以直接在 DOS 命令行中进行开发、调试、编译、打包、部署等工作，由于这种开发效率太低，Android 提供了针对 Eclipse 的开发插件 ADT（Android

Development Tools）。ADT 极大地提高了开发效率，可以在 Eclipse 中快速创建 Android 应用程序，自动生成一些代码。

1.3　SDK 与 ADT 的下载和配置

本节将讲述 Android 开发环境的搭建，以及模拟器（ADT）的创建。ADT 必须有 Eclipse 和 Android SDK 的支持，Eclipse 必须有 JDK 的支持。安装环境的正确配置是：下载 Android SDK、下载并安装 JDK、下载 Eclipse、下载并安装 ADT。Android 开发环境的搭建，需要如下软件开发包。

- ❑ JDK 请到网站：http://www.oracle.com/technetwork/java/javase/downloads/index.html 处下载。
- ❑ Eclipse 请到网站：http://www.eclipse.org/downloads 处下载。
- ❑ Android SDK 请到网站：http://developer.android.com 处下载。
- ❑ ADT 请到网站：http://developer.android.com/sdk/eclipse-adt.html 处下载。

接下来我们以 MyEclipse 8.5 及 ADT-8.0.1 为例，详细讲解如何配置 ADT。

（1）首先解压 ADT-8.0.1.zip 压缩文件。把 plugins 目录下的 jar 文件放到 Genuitek/Common/plugins 目录下，把 features 目录下的 jar 文件解压放在 Genuitek/Common/features 目录下，然后修改 MyEclipse 8.5/configuration/org.eclipse.equinox.simpleconfigurator 下的 bundles.info 文件，加入下面三行代码：

```
com.android.ide.eclipse.adt,8.0.1.v201012062107-82219,file:/D:/tools/
ProgramFiles/MyEclipse8.5/Common/plugins/com.android.ide.eclipse.adt
8.0.1.v201012062107-82219.jar,4,false

com.android.ide.eclipse.ddms,8.0.1.v201012062107-82219,file:/D:/tools/
ProgramFiles/MyEclipse8.5/Common/plugins/com.android.ide.eclipse.ddms
8.0.1.v201012062107-82219.jar,4,false

com.android.ide.eclipse.hierarchyviewer,8.0.1.v201012062107-82219,file:
/D:/tools/ProgramFiles/MyEclipse8.5/Common/plugins/com.android.ide.ecli
pse.hierarchyviewer_8.0.1.v201012062107-82219.jar,4,false
```

（2）完成如上所述步骤，然后重启 MyEclipse 8.5。选择 Window|Preferences 命令，在弹出的 Preferences 窗口左侧多了一项“Android”，选择 Android 选项，在右边的对话框中，为 SDK Location 选项选择 Android SDK 的路径，下面会列出当前可用的 SDK 版本和 Google API 版本，如图 1.1 和图 1.2 所示。

（3）接下来就是创建一个模拟器（AVD），选择 Window|Android SDK and AVD Manager 命令。在弹出的 Android SDK and AVD Manager 窗口右侧单击“New...”按钮，弹出 Create new Android Virtual Device（AVD）窗口，如图 1.3 所示。在 Name 右侧的输入框输入要创建的模拟器名称；在 Target 右侧的选择框中选择 API 版本；Size 右侧的输入框是输入模拟器 SD Card 大小；也可以选择下面的 File 选项；下面几项就选择默认的就行，最后单击 Create AVD 按钮，模拟器（AVD）创建成功，列表中列出创建的模拟器，如图 1.4 所示。

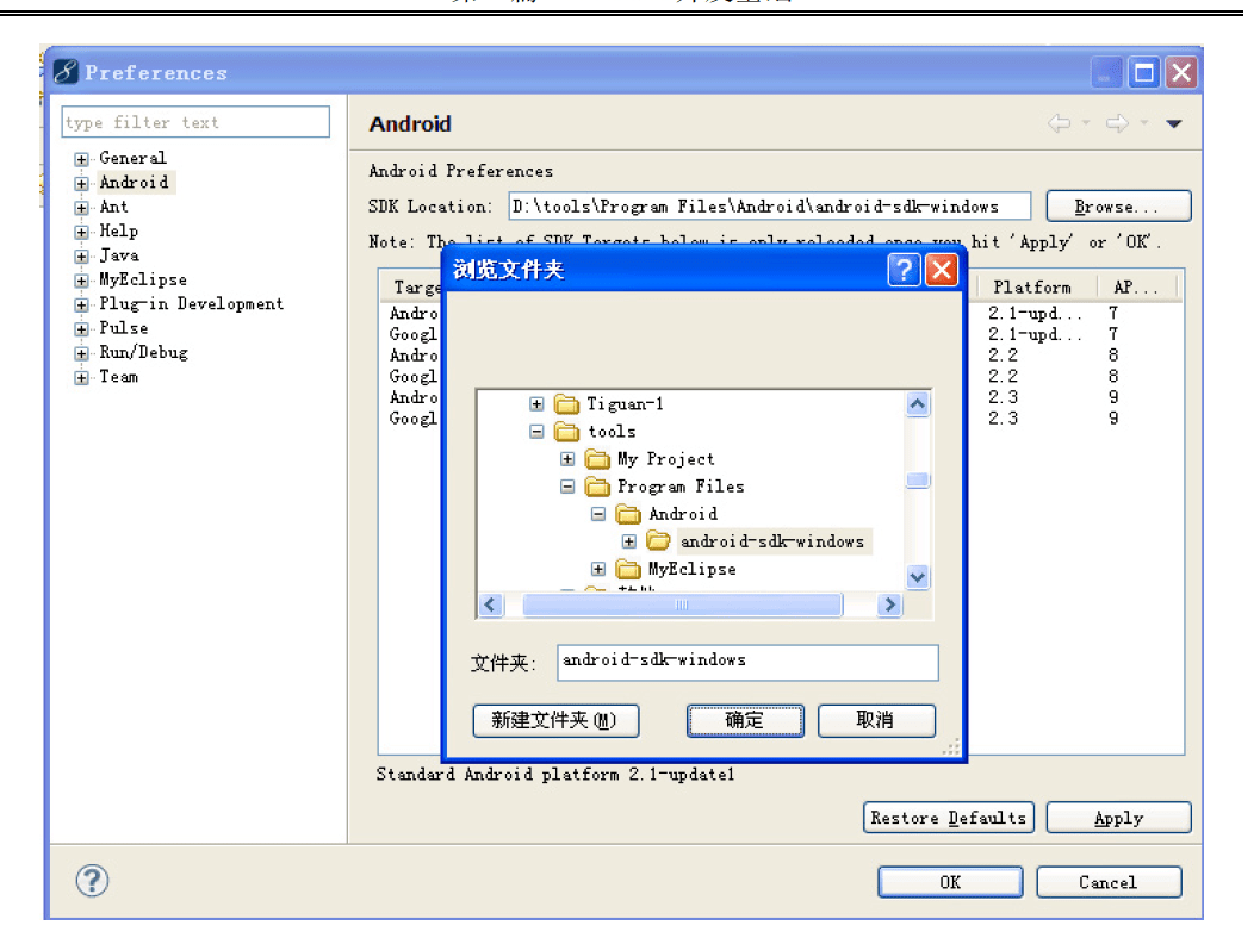


图 1.1　Android SDK 配置图

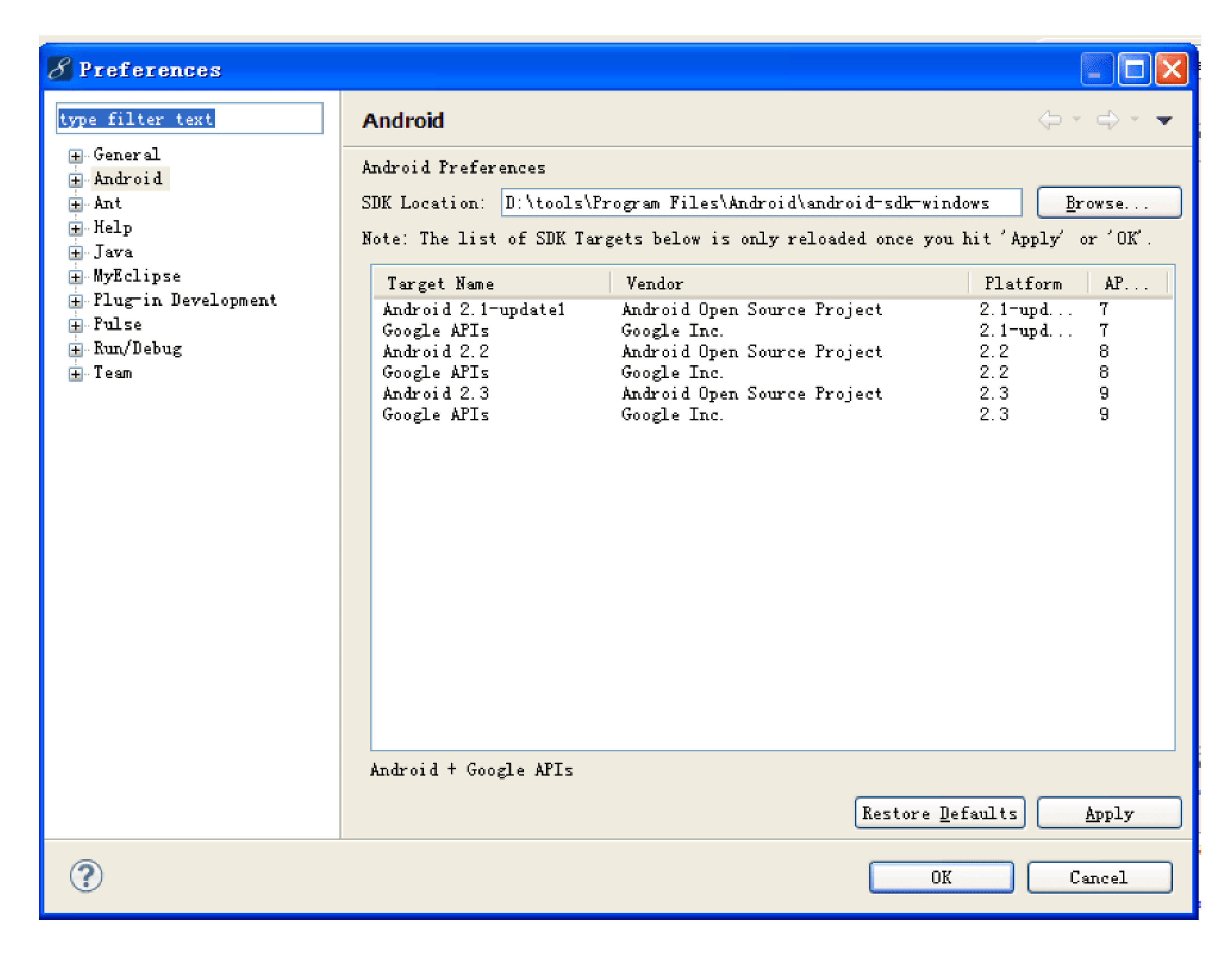


图 1.2　Android SDK 配置成功图

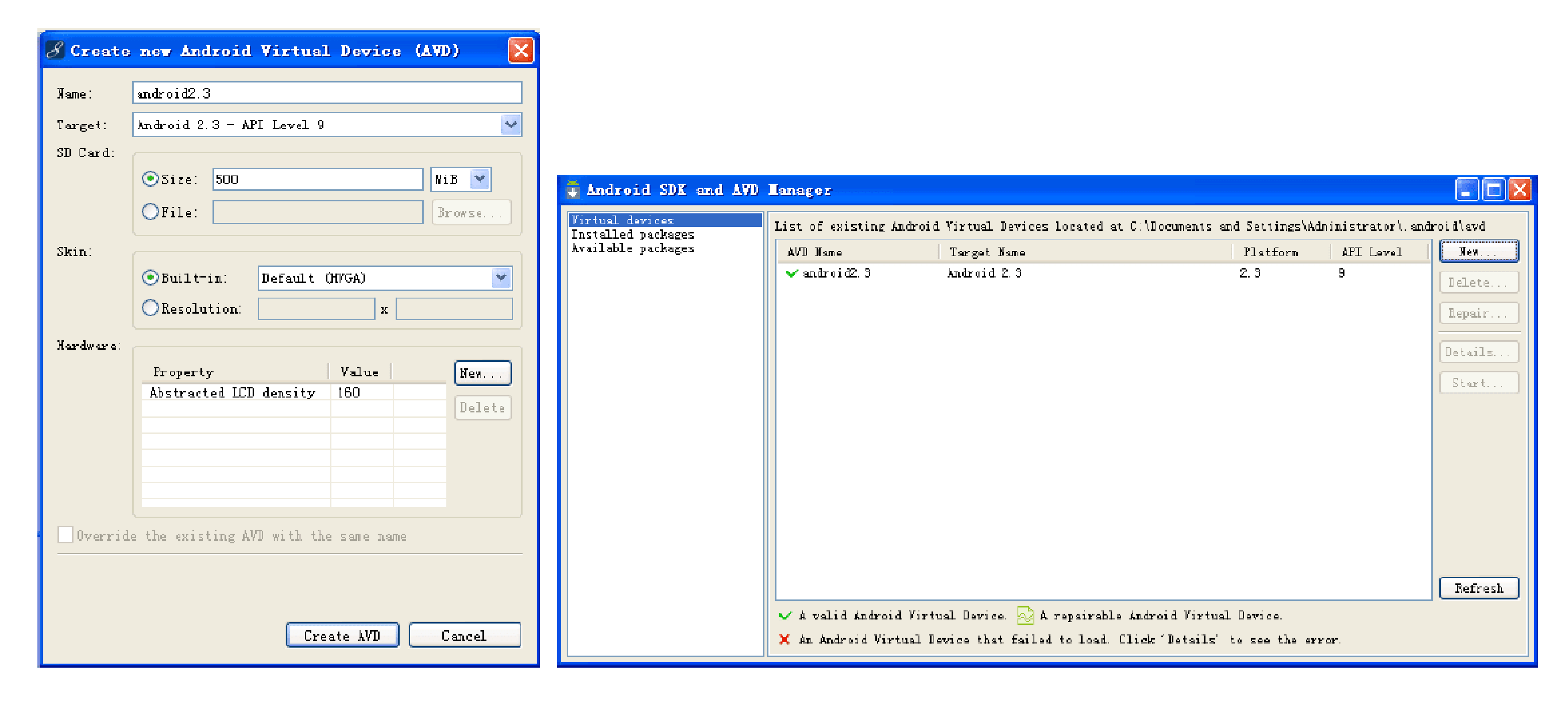


图 1.3　创建 AVD 图　　　　图 1.4　AVD 创建成功图

完成以上步骤，我们的模拟器（AVD）就创建好了。在实际开发中，模拟器（AVD）可以说必不可少，所以学会创建模拟器（AVD）就是重要的一环。下一节我们将真正地接触如何写一个 Android 应用程序。

1.4　创建第一个 Android 项目 “Hello World”

通过前面几节的学习我们了解了 Android 和怎样搭建相应的开发环境。这一节，我们来动手创建我们的第一个 Android 项目 “Hello World” 。

（1）选择 File | New | Project 命令，弹出 New Project 窗口，如图 1.5 所示。选择 Android | Android Project 命令，单击 Next 按钮进行下一步骤。

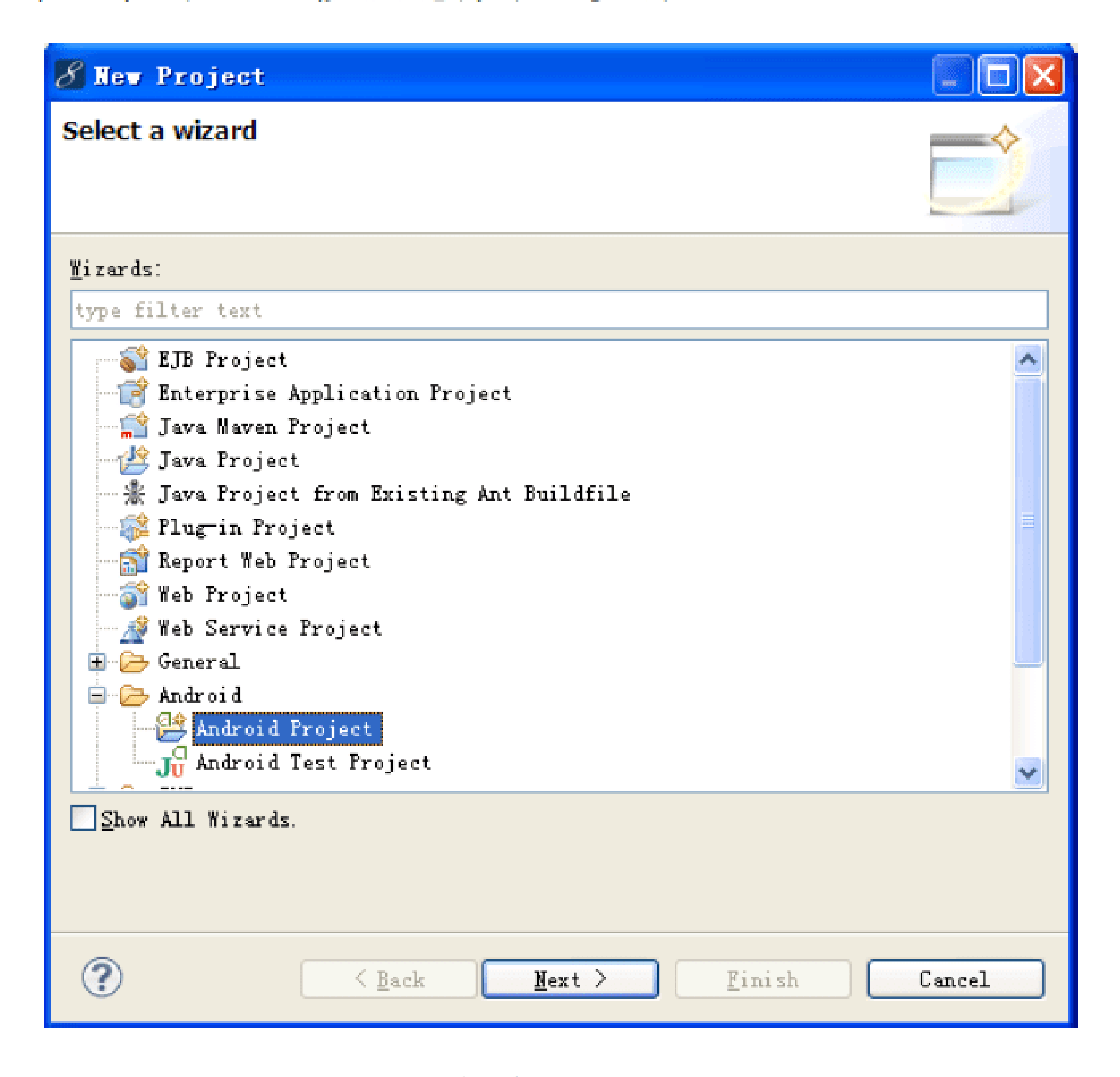


图 1.5　新建 Android 项目图

（2）弹出 New Android Project 窗口，如图 1.6 所示。在 Project name 右侧的输入框中输入项目名称“HelloWorld”；下面的“Create new project in workspace”选项是指选择默认的工作区，“Create project from existing source”选项是指自定义工作区，这里我们选择默认的工作区。

（3）在 Build Target 列表中选择要创建的项目 API 版本，这里我们选择 Android 2.3；在 Application name 右侧的输入框中输入应用程序的名称“Hello World”。在 Package name 右侧的输入框中输入包名“com.hello”。在 Create Activity 右侧的输入框中输入要创建 Activity 名称“HelloActivity”。在 Min SDK Version 右侧的输入框中输入最小 SDK 层级“9”。单击 Finish 按钮，完成项目创建。

完成以上步骤，我们就创建了一个名为“HelloWorld”的项目，项目结构如图 1.7 所示。

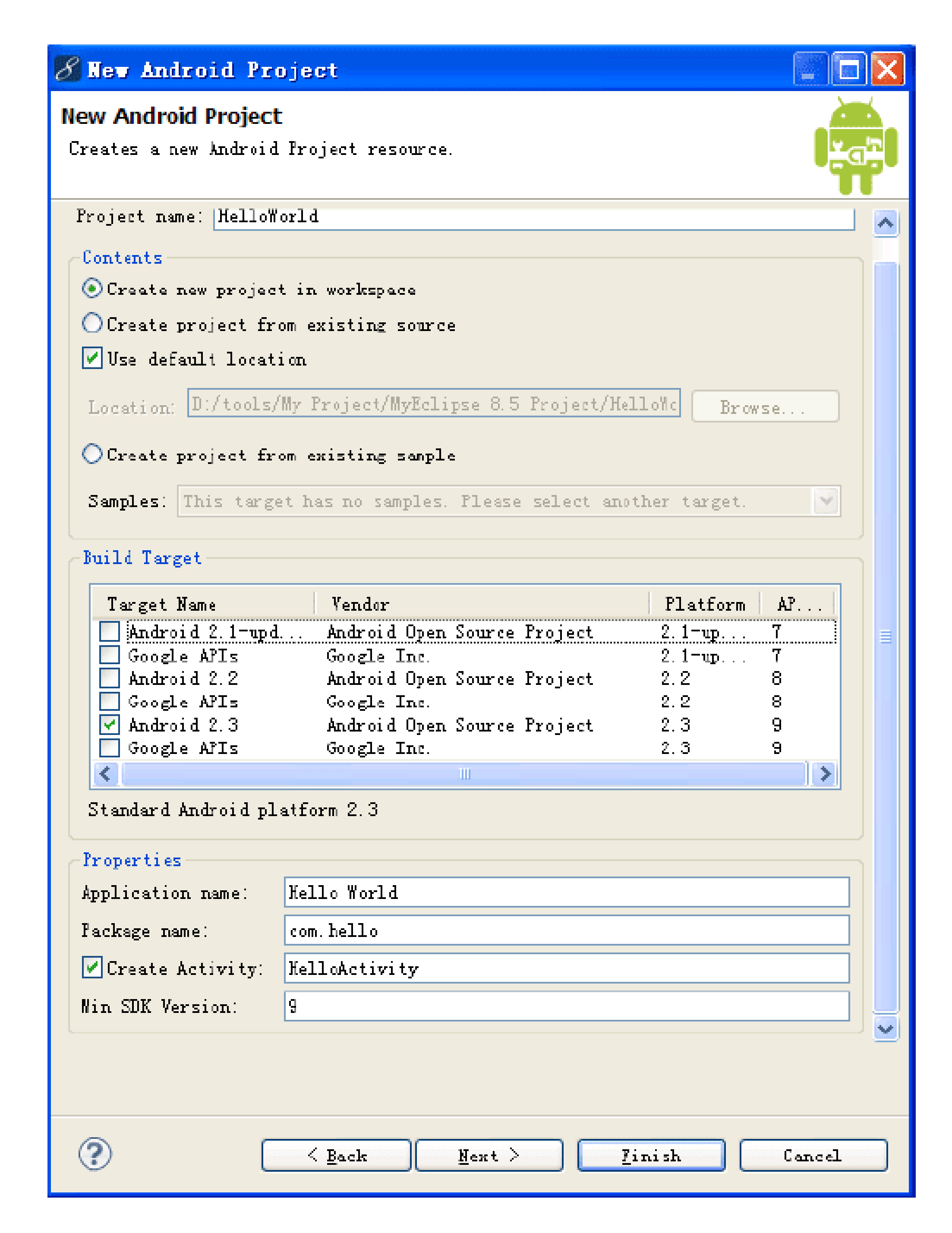



图 1.6　Android 项目创建图

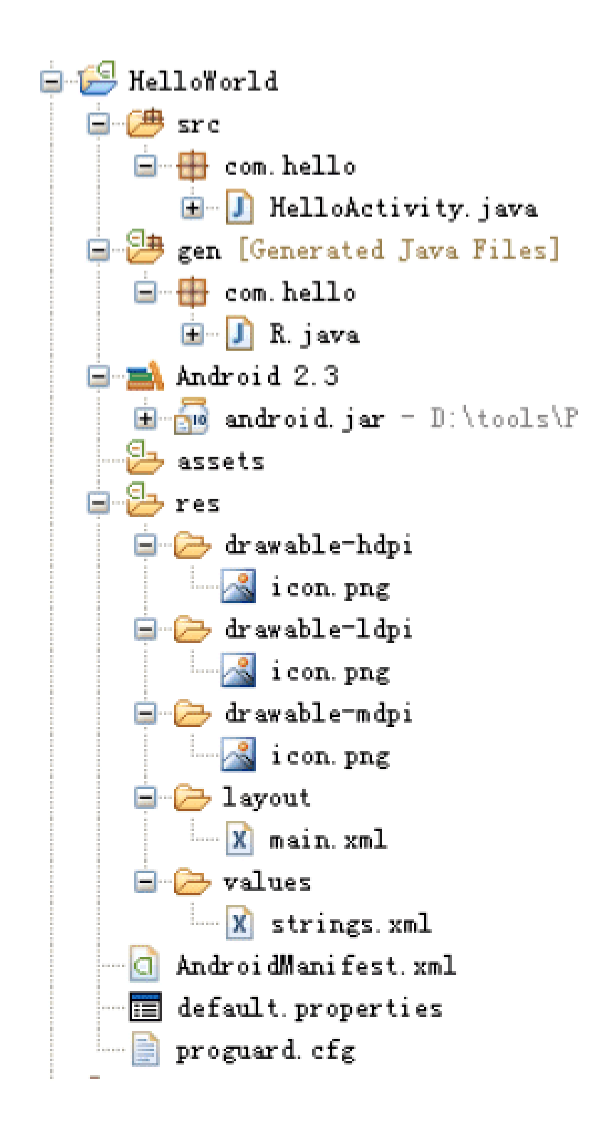


图 1.7　Android 项目结构图

（4）接下来编写这个“HelloWorld”的项目，代码如下。

HelloActivity.java

```
package com.hello;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class HelloActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);          //设置 Activity 的显示界面
    }
}
```

HelloActivity.java 是 Android 应用的 Activity 类，该类继承 Activity。覆盖父类的 onCreate()方法。

（5）在图 1.7 中，main.xml 是界面布局文件，代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#FFFFFF"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    android:textColor="#000000"
    />
</LinearLayout>
```

这段代码的含义如下。

- <LinearLayout>：流式布局元素标签，每一个 XML 元素提供大量的属性，用于修改元素的外观。
- xmlns:android：LinearLayout 元素的属性，指定 XML 的命名空间，告诉 Android 的开发工具使用该命名空间中的元素和属性（所有的 Android 设计文件必须引入这个命名空间）。
- android:orientation：该属性用于指定流式布局的方向，流式布局有两种布局方向，分别为“vertical”表示垂直布局，“horizontal”表示水平布局。
- android:layout_width：该属性用于指定元素的水平宽度，有 3 种取值，分别为“fill_parent”表示当前元素的宽度由父元素的宽度决定，即会填充父元素，该值在 Android 2.2 开始由“match_parent”所取代，他们的定义本质都是一样的，值均为-1，只是换了个别名。“wrap_content”表示当前元素的宽度由元素本身的内容决定，即根据内容的不同自动调节元素的宽度。例如，android:layout_width=“100px”或者 android:layout_width=“100dip”。
- android:layout_height：该属性用于指定元素的垂直高度，取值和 android:layout_width 取值相同。
- android:background：该属性用于设置元素的背景颜色。
- <TextView>：文本显示元素标签，所有用于显示的信息都可以通过该元素实现。
- android:text：该属性指定元素上要显示的信息。
- android:textColor：该属性用于设置文本字体的颜色。

（6）在图 1.7 中，strings.xml 是配置文件，该文件用于声明一些在程序中使用的字符串常量。代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World!</string>
    <string name="app_name">Hello World</string>
</resources>
```

我们现阶段只需编写如上文件的代码即可，是不是很容易？接下来我们看看运行效果，如图 1.8 所示。

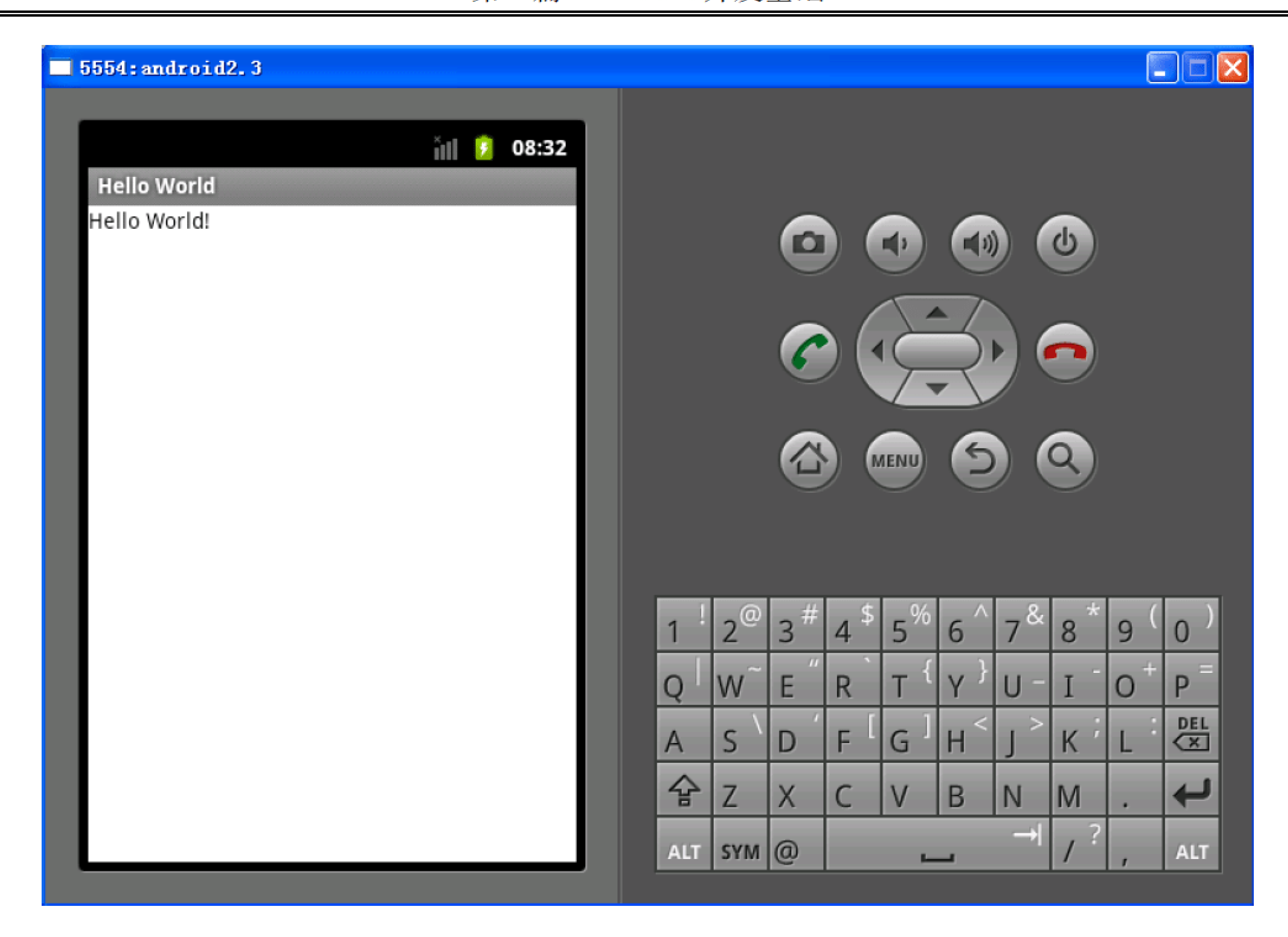


图 1.8　“HelloWorld”效果图

第 2 章　Android 工程组件介绍

说到 Android 应用开发，就一定要涉及 Android 组件。所谓的组件（Component），大家可以理解为装修房屋的一个组成元素，一个 Android 应用就像一间房子，房子里的一张桌子、一把椅子、一张床就相当于 Android 的组件。除了这种可以直观上看得到的组件外，还有一些组件负责服务功能，例如房子中的水管道、燃气管道、电力管道等，这些不是直接呈现在视觉中，但却起着很重要的作用，就相当于 Android 中的 Service 组件。当你要实现一个 Android 应用的时候，就可以把一堆接口标准、封装完整的组件拿来用，组件搭配使用，就形成了一间装修好的房子，也就是一个 Android 应用了。

Android 是一个为组件化而搭建的平台。具体说，有 4 大组件：Activity、Service、BroadcastReceiver 和 Content Provider。

2.1　Activity 介绍

Activity 是 Android 应用程序与用户交互的窗口，几乎每一个 Android 应用程序都离不开 Activity。Activity 就像一个网站的页面一样，例如你的应用中，可以有登录页面、注册页面、评论页面和产品列表页面，每一个页面都可以通过一个独立的类来表示。这个独立的类继承于 Activity 这个基类，上面可以显示由几个 View 组件组成的用户接口，并且可以对事件进行相应的处理，一套 View 通过 Activity.setContentView()填充到 Activity 窗体中。每当应用中添加一个 Activity，需要在 AndroidManifest.xml 中添加一个对应的<activity></activity>标签，可以设置某一个 Activity 为第一个显示的窗体。

2.1.1　Activity 的生命周期

系统中的 Activity 可以通过一个 Activity 栈来进行管理。当一个新的 Activity 启动的时候，它会首先被放置在 Activity 栈顶部，并且该 Activity 的状态为 Running，之前的 Activity 也在 Activity 栈中，但是被保存在它的下边，只有当这个新的 Activity 退出后，以前的 Activity 才能重新回到前景界面。

Activity 有以下 4 种基本状态。

- ❑ Active/Running 状态：一个新的 Activity 启动入栈后，它位于屏幕的最前端，Activity 栈的最顶端，此时它处于可见并可以和用户交互的状态。
- ❑ Paused 状态：Activity 失去了焦点，但仍然可见，这时为 Paused 状态。例如，被另一个透明的或者 Dialog 样式的 Activity 覆盖时的状态，此时它依然与窗口管理器保持连接，系统依然继续维护它的内部状态。因为失去了焦点，故不可与用户

交互。

- ❑ Stopped 状态：当 Activity 被另一个 Activity 完全覆盖、失去焦点，并不可见，这时处于 Stopped 状态。它仍然保持着所有的状态和成员信息，但对于用户来讲，它是不可见的。当系统内存需要被用在其他地方的时候，Stopped 的 Activity 将被杀掉。
- ❑ Killed Activity 状态：被系统杀死回收或者没有被启动时处于 Killed 状态。Activity 为 Paused 或者 Stopped 状态，系统需要清理内存时，可以通过 finish 或者 kill 结束其进程。当需要重新显示时，必须完全重新启动，并将其关闭之前的状态全部恢复。

当一个 Activity 实例被创建、销毁或者启动另一个 Activity 时，这四种状态之间会进行切换，这种切换取决于用户程序的动作。如图 2.1 所示，展示了 Activity 在不同状态间转换的状态图及转换条件。

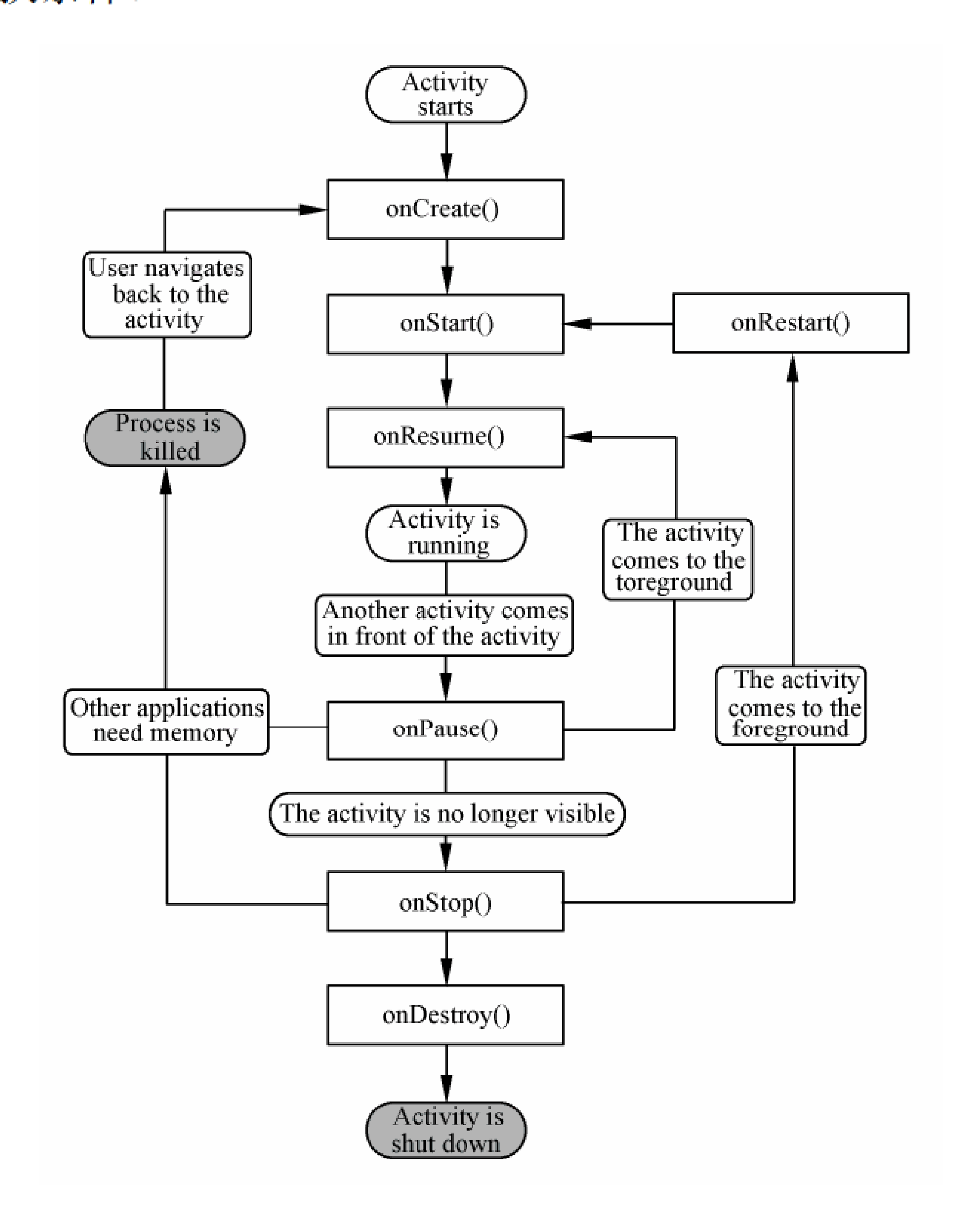


图 2.1　Activity 状态转换图

如上图所示，标记颜色的部分为 Activity 的主要生命周期，Activity 有 3 个关键的生命周期，即完整生命周期、前景生命周期和可见生命周期。

- ❑ 完整生命周期：从调用 onCreate()方法到最终调用 onDestroy()方法称为完整生命周期。Activity 会在 onCreate()方法进行所有“全局”状态的设置，在 onDestroy()方法中释放所有持有的资源。
- ❑ 可见生命周期：从调用 onStart()方法开始，到调用 onStop()方法为止称为可见生命周期。这期间用户可以在屏幕上看见这个 Activity，但并不一定是在前景，也不一定可以和用户交互。这两个方法之间可以维护 Activity 在用户显示时所需要的

资源。

- 前景生命周期：从调用 onResume()方法开始，到调用 onPause()方法为止称为前景生命周期。这段时间 Activity 处于其他所有 Activity 的前面，且与用户交互。一个 Activity 可以经常从 Resumed 和 Paused 状态之间转换，如新的 Intent 到来时或者手机进入休眠状态时。

下面的 Activity 方法定义了 Activity 完整的生命周期，你可以重写这些方法达到在 Activity 状态改变时执行你所期望的操作。所有的 Activity 都应该实现自己的 onCreate()方法进行初始化设置，大部分还应该实现 onPause()方法提交数据的修改及准备终止和用户的交互。大多数的 Activity 还需要实现 onFreeze()方法并在 onCreate()方法中执行对应状态恢复。其他方法可以在需要时进行设置，当实现这些方法时，需要注意的是一定要调用父类中的对应方法。

```
public class Activity extends ApplicationContext {
    protected void onCreate(Bundle icicle);
    protected void onStart();
    protected void onRestart();
    protected void onResume();
    protected void onFreeze(Bundle outIcicle);
    protected void onPause();
    protected void onStop();
    protected void onDestroy();
 }
```

以上方法的含义说明如下。

- protected void onCreate(Bundle icicle)：Activity 初次创建时调用该方法。一般情况下，我们会重写该方法，并作为应用程序的入口，在这个方法中可以进行初始化数据、设置用户界面等操作。大多数情况下，我们需要在这里加载用户界面 XML 资源文件，例如 setContentView(R.layout.main)，如果 Activity 之前存在冻结状态，那么此状态将在 Bundle 中提供，如果 Activity 首次创建，本方法后将会调用 onStart()方法，如果 Activity 是停止后重新显示，则将调用 onRestart()方法。
- protected void onStart()：该方法在 onCreate()方法之后被调用，或者在 Activity 从 Stop 状态转换为 Active 状态时被调用。其后调用 onRestart()方法或者 onResume()方法。
- protected void onRestart()：当 Activity 从停止状态重新启动时调用。其后调用 onResume()方法。
- protected void onResume()：当 Activity 将要与用户交互时调用此方法，此时 Activity 在栈顶，用户输入已经可以传递给它。如果其他的 Activity 在它的上方恢复显示，则将调用 onFreeze()方法。
- protected void onFreeze(Bundle outIcicle)：当 Activity 暂停，其他的 Activity 恢复与用户交互的时候调用这个方法。
- protected void onPause()：当系统要调用其他的 Activity 时调用（其他 Activity 显示之前），一般该方法用来提交数据的改变，停止动画，和其他占用 CPU 资源的东西。如果 Activity 重新回到前景则调用 onResume()方法，如果对用户彻底不可见则调用 onStop()方法。

- ❑ protected void onStop()：当其他 Activity 恢复并遮盖此 Activity，导致此 Activity 对用户不可见时调用。当 Activity 重新回到前景与用户交互时调用 onRestart()方法，如果 Activity 将退出则调用 onDestroy()方法。
- ❑ protected void onDestroy()：在 Activity 被销毁前调用的最后一个方法，当进程终止时会出现这种情况（调用 Activity 提供一个的 finish()方法或者系统为了节省空间而临时销毁 Activity 的实例，可以通过 isFinishing()方法返回值区分这两种情况）。

下面通过例子演示 Activity 的生命周期，以及各个方法的调用情况。ActivityLife.java 文件代码如下：

```
package com.ActivityLifeExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ActivityLife extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);     //加载资源文件 frist.xml
        System.out.println("onCreate");    //该输出信息会在 LogCat 中看到
    }

    @Override
    protected void onDestroy() {//在 Activity 被销毁前调用的最后一个方法
        //TODO Auto-generated method stub
        super.onDestroy();
        System.out.println("onDestroy");
                                //用于测试的语句，在 LogCat 中可以看到该输出信息
    }

    @Override
    protected void onRestart() {    //当 Activity 从停止状态重新启动时调用
        //TODO Auto-generated method stub
        super.onRestart();
        System.out.println("onRestart");
                                //用于测试的语句，在 LogCat 中可以看到该输出信息
    }

    @Override
    protected void onResume() {//当 Activity 将要与用户交互时调用此方法
        //TODO Auto-generated method stub
        super.onResume();
        System.out.println("onResume");
                                //用于测试的语句，在 LogCat 中可以看到该输出信息
    }

    //该方法在 onCreate()方法之后被调用，或者在 Activity 从 Stop 状态转换为 Active
    状态时被调用
    @Override
    protected void onStart() {
        //TODO Auto-generated method stub
        super.onStart();
        System.out.println("onStart");
                                //用于测试的语句，在 LogCat 中可以看到该输出信息
    }
    //当其他 Activity 恢复并遮盖此 Activity，导致此 Activity 对用户不可见时调用
```

```
    @Override
    protected void onStop() {
        //TODO Auto-generated method stub
        super.onStop();
        System.out.println("onStop");
                                //用于测试的语句，在 LogCat 中可以看到该输出信息
    }

    @Override
    protected void onPause() { //当系统要调用其他的 Activity 时调用
        //TODO Auto-generated method stub
        super.onPause();
        System.out.println("onPause");
                                //用于测试的语句，在 LogCat 中可以看到该输出信息
    }
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- <LinearLayout >：流式布局标签，关于布局详细内容将在第 3 章介绍-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout height="fill parent">
    <!--<TextVIew>：显示文本信息标签，关于 Android 的详细标签及组件介绍将在第 4 章
    介绍-->
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Activity 生命周
        期测试" />
</LinearLayout>
```

AndroidMainfest.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ActivityLifeExample" android:versionCode="1"
    android:versionName="1.0">
    <!--<application>：Android 应用标签。属性 android：icon：表示应用的图标。
        android：label：应用的名称。-->
    <application android:icon="@drawable/icon" android:label="@string/
    app name">
        <!-- <activity>：窗体标签。属性 android：name：窗体类名。
             Android：label：窗体标题栏上显示的文字-->
        <activity android:name=".ActivityLife" android:label="@string/
        app name">
            <intent-filter>
                <!—设置该 Activity 为启动窗体-->
                <action android:name="android.intent.action.MAIN" />
                <!—设置该 Activity 在应用列表中显示-->
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

启动该 Activity，在 LogCat 窗口观察方法执行的先后次序及方法执行的条件，当打开应用后执行了 onCreate()、onStart()和 onResume()3 个方法，LogCat 视窗如图 2.2 所示。

04-30 11:36:21.587	I	373	System.out	onCreate
04-30 11:36:21.607	I	373	System.out	onStart
04-30 11:36:21.607	I	373	System.out	onResume

图 2.2 LogCat 窗口输出信息

当按 Back 键后，该应用程序将结束，这个时候将先后调用 onPause()、onStop()和 onDestroy()3 个方法，如图 2.3 所示。

04-30 11:40:20.836	I	373	System.out	onPause
04-30 11:40:21.567	I	373	System.out	onStop
04-30 11:40:21.577	I	373	System.out	onDestroy

图 2.3 LogCat 窗口输出信息

按 Back 键，退出应用程序后，再重新启动 Activity，运行效果和图 2.2 相同。

当我们打开应用程序时，又想浏览一下网页，或者看一下电子书，这时候我们会选择按 Home 键，然后去打开电子书应用程序或者浏览器。当我们按下 Home 的时候，Activity 先后执行了 onPause()和 onStop()方法，这时候应用程序并没有销毁，如图 2.4 所示。

04-30 11:48:36.076	I	373	System.out	onPause
04-30 11:48:36.766	I	373	System.out	onStop

图 2.4 LogCat 窗口输出信息

当我们再次启动该应用程序时，则先后执行的方法为 onRestart()、onStart()和 onResume()，如图 2.5 所示。

04-30 11:50:05.486	I	373	System.out	onRestart
04-30 11:50:05.486	I	373	System.out	onStart
04-30 11:50:05.486	I	373	System.out	onResume

图 2.5 LogCat 窗口输出信息

看完这个例子，大家再回过头看图 2.1 所示的 Activity 生命周期及状态切换就很容易理解了。

2.1.2 调用另一个 Activity—Intent 的使用

Android 组件之间的通信，由 Intent 协助完成。Intent 负责对应用中的操作动作，及动作涉及的数据进行描述，Android 系统根据该 Intent 的描述，找到对应的组件，将 Intent 传递给调用的组件。Activity 提供了 startActivity(Intent intent)方法，用来调用另一个 Activity，参数为 Intent 类型，在 Intent 类中提供了 setClass()方法可以指定跳转的目标 Activity 是哪一个。Intent 不仅可以用于应用程序，也可以用于应用程序内部的 Activity/Service 之间的交互。

下面我们通过例子，看看如何通过一个 Activity 跳到另一个 Activity。

ActivityExample1.java 类的 Java 代码如下：

```
package com.ActivityExample1;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ActivityExample1 extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);          //加载资源文件 main.xml
        Button button=(Button) this.findViewById(R.id.but1);
                                                //从资源文件中获取 Button 对象
        button.setOnClickListener(new OnClickListener(){
                                                //监听 Button 对象的单击事件
             @Override
             public void onClick(View v) {
                 //TODO Auto-generated method stub
                 Intent intent=new Intent();    //创建 Intent 对象
                 //setClass():指定要启动的 Activity
                 intent.setClass(ActivityExample1.this, SecondActivity.
                 class);
                 startActivity(intent);         //启动新的窗体
             }
         });
     }
}
```

SecondActivity.java 的 Java 代码如下：

```
package com.ActivityExample1;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class SecondActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);    //加载布局资源文件 second.xml
    }
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- <LinearLayout >: 流式布局标签，关于布局详细内容将在第 3 章介绍-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout height="fill parent" android:background="#FFFFFFFF">
     <!--<TextVIew>: 显示文本信息标签，关于 Android 的详细标签及组件介绍将在第 4 章
     介绍-->
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="第一个 Activity"
        android:textColor="#FF000000" />
     <!--< Button>: 按钮标签，关于 Android 的详细标签及组件介绍将在第 4 章介绍-->
    <Button android:id="@+id/but1" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="跳转到第二个
        Activity" />
</LinearLayout>
```

Res/layout/second.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<!-- <LinearLayout >：流式布局标签，关于布局详细内容将在第 3 章介绍-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFFFFFFF">
     <!--<TextVIew>：显示文本信息标签，关于 Android 的详细标签及组件介绍将在第 4 章
     介绍-->
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="第二个 Activity"
        android:textColor="#FF000000" />
</LinearLayout>
```

在本例中创建了两个 Activity 的类，需要在 AndroidManifest.xml 文件中声明每一个 Activity。AndroidManifest.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ActivityExample1" android:versionCode="1"
    android:versionName="1.0">
     <!--<application>：Android 应用标签。属性 android：icon：表示应用的图标。
          android：label：应用的名称。-->
    <application android:icon="@drawable/icon" android:label="@string/
    app_name">
        <!-- <activity>：窗体标签。属性 android：name：窗体类名。
              Android：label：窗体标题栏上显示的文字-->
        <activity android:name=".ActivityExample1" android:label="@string/
        app_name">
            <intent-filter>
                 <!--设置该 Activity 为启动窗体-->
                <action android:name="android.intent.action.MAIN" />
                 <!--设置该 Activity 在应用列表中显示-->
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SecondActivity" android:label="@string/
        app_name"/>
    </application>
</manifest>
```

以上代码的运行结果如图 2.6 所示。

单击窗体上的按钮，跳转到第二个 Activity，如图 2.7 所示。

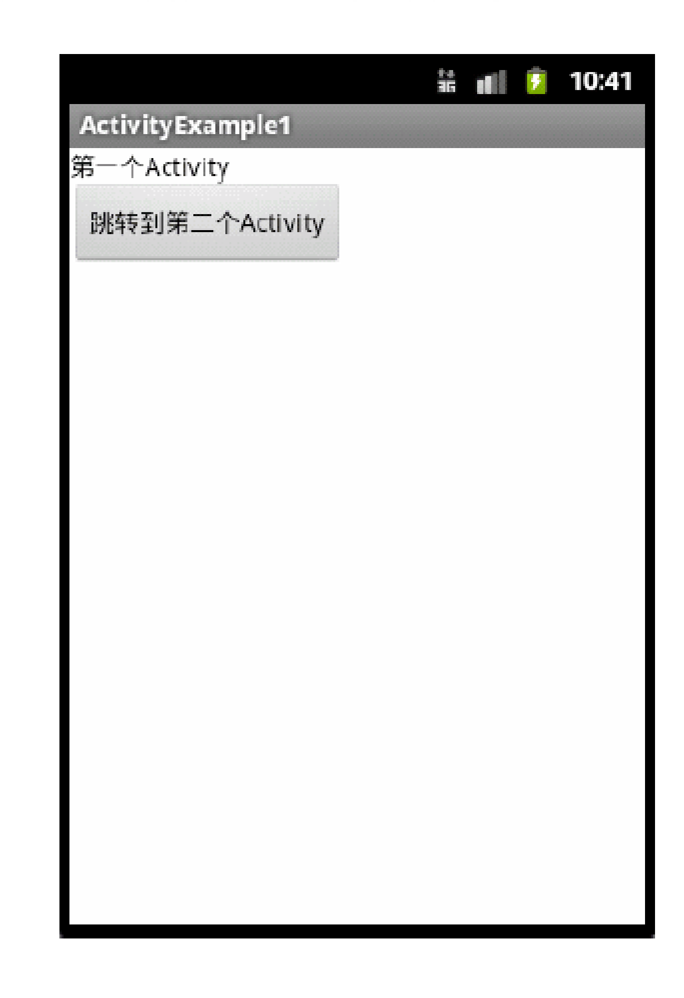


图 2.6　Activity 之间的跳转

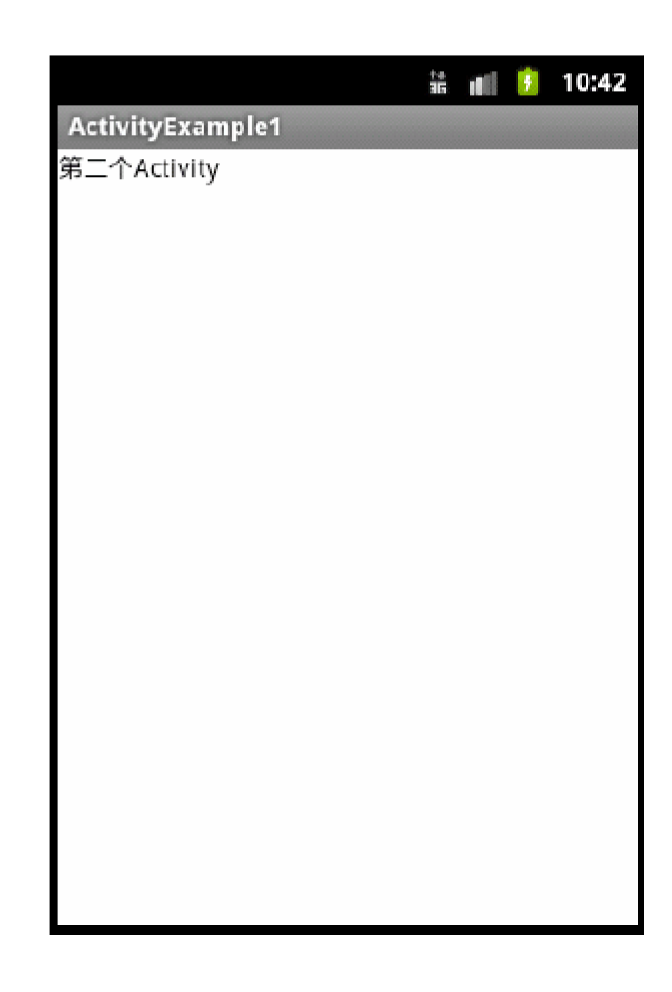


图 2.7　Activity 之间的跳转

2.1.3　使用 Bundle 在 Activity 间传递数据

Bundle 用来实现 Activity 之间的数据传递，Bundle 相当于 Map 类，即通过（Key，Value）方式描述数据，用 Bundle 绑定数据，便于数据的处理。Bundle 中提供了 putXXX()和 getXXX()方法用来保存和获取数据。下面的例子模拟了一个评论功能，在第一个窗体中输入评论信息，单击“提交”按钮显示在第二个窗体上。

ActivityExample2.java 文件的代码如下：

```
package com.ActivityExample2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ActivityExample2 extends Activity {
    /** Called when the activity is first created. */
     private EditText myText;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);       //加载资源文件 main.xml
        myText=(EditText) this.findViewById(R.id.content);
                                             //从资源文件中获取 EditText 对象
        Button button=(Button) this.findViewById(R.id.but1);
                                             //从资源文件中获取 Button 对象
        button.setOnClickListener(new OnClickListener(){
                                             //监听 Button 对象的单击事件
              @Override
              public void onClick(View v) {
                  //TODO Auto-generated method stub
                  String str=myText.getText().toString();
                                                  //获取 EditText 中显示的内容
                  Intent intent=new Intent();     //创建 Intent 对象
                  Bundle bundle = new Bundle();   //实例化 Bundle
                  //setClass():指定要启动的 Activity
                  intent.setClass(ActivityExample2.this, SecondActivity.
                  class);
                  bundle.putString("message", str);
                                             //向 Bundle 中添加 String 类型数据
                  intent.putExtras(bundle);  //为 Intent 添加数据
                  startActivity(intent);     //启动新的窗体
              }
        });
    }
}
```

SecondActivity.java 代码如下：

```
package com.ActivityExample2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class SecondActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);     //加载布局资源文件 second.xml
        Bundle bundle=this.getIntent().getExtras();
                                             //从 Intent 中获取传递过来的数据
```

```
        //获取 key 为 message 的数据及传递过来的评论内容数据
        String message=bundle.getString("message");
        TextView myText=(TextView) this.findViewById(R.id.showMessage);
                                            //从资源文件中获取 TextView 组件
        myText.setText(message);            //将评论内容显示在 TextView 中
    }
}
```

SecondActivity.java 代码如下：

```
package com.ActivityExample2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class SecondActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);     //加载布局资源文件 second.xml
        Bundle bundle=this.getIntent().getExtras();
                                            //从 Intent 中获取传递过来的数据
        //获取 key 为 message 的数据及传递过来的评论内容数据
        String message=bundle.getString("message");
        TextView myText=(TextView) this.findViewById(R.id.showMessage);
                                            //从资源文件中获取 TextView 组件
        myText.setText(message);            //将评论内容显示在 TextView 中
    }
}
```

Res/layout/second.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- <LinearLayout >: 流式布局标签，关于布局详细内容将在第 3 章介绍-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout height="fill parent" android:background="#FFFFFFFF">
     <!--<TextVIew>: 显示文本信息标签，关于 Android 的详细标签及组件介绍将在第 4 章
     介绍-->
    <TextView android:id="@+id/showMessage" android:layout width="fill
    parent"
        android:layout height="wrap content" android:textColor="#FF000000" />
</LinearLayout>
```

运行结果如图 2.8 所示。

单击“提交“按钮，进入下一个窗体，如图 2.9 所示。



图 2.8　Activity 之间参数传递

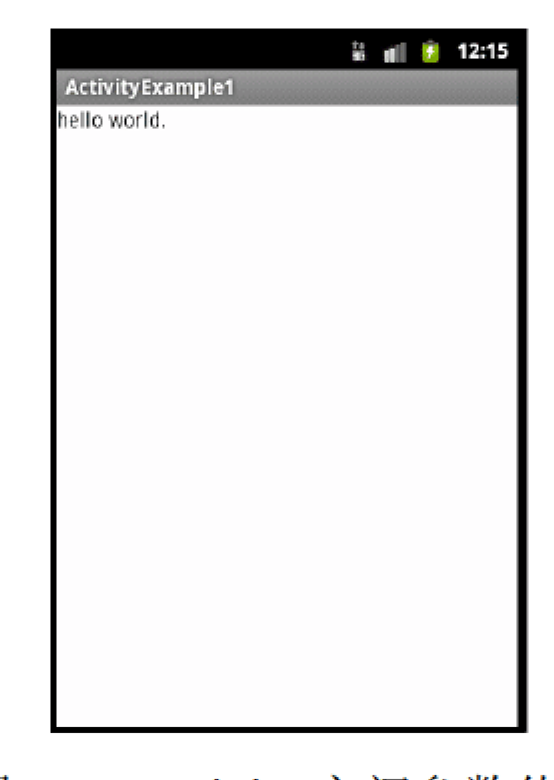


图 2.9　Activity 之间参数传递

2.2　Service 介绍

Service 没有用户界面，运行在后台，负责处理一些用户看不到、并且会有持续时间的事情。如果我们退出应用，Service 进程并没有结束，它仍然在后台运行，一般在播放音乐，下载数据等情况下会用到 Service。有时候我们想一边听音乐，一边做些其他的事情，当我们退出音乐应用时，如果不用 Service，我们就听不到音乐了，这时候 Service 就发挥它的作用了。自己创建的 Service 需要继承 android.app.Service 类，并且要在 AndroidManifest.xml 文件中通过<service>标签注册。可以通过 startService()或 bindService()方法启动 service，通过 stopService()方法或者 unBindService()方法停止 Service。

另外注意 Service 是跑在程序主线程中的，处理耗时事情需要启动一个线程，以防止阻塞主线程。Service 只需要重写 3 个方法，即 onCreate()、onStart()和 onDestroy()方法。当第一次启动 Service 时，先后调用了 onCreate()和 onStart()方法；当停止 Service 时，执行 onDestroy()方法；当 Service 为启动状态时，再启动 Service，不会执行 onCreate()方法，而直接执行 onStart()方法。

下面自定义 Myservice 类继承 Service，通过 ServiceExample 类测试 Service 的启动及停止。

MyService.java 代码如下：

```
package com.ServiceExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class MyService extends Service {
    private MyBinder myBinder=new MyBinder();
    @Override
    public IBinder onBind(Intent intent) {
        //TODO Auto-generated method stub
        System.out.println("start IBinder");
        return myBinder;
    }
    public void onCreate(){
        System.out.println("start onCreate");
        super.onCreate();
    }
    public void onStart(Intent intent,int startId){
        System.out.println("start onStart");
        super.onStart(intent, startId);
    }
    public void onDestroy(){
        System.out.println("start onDestroy");
        super.onDestroy();
    }
    public boolean onUnbind(Intent intent){
        System.out.println("start onUnbind");
        return super.onUnbind(intent);
    }
    public String getSystemTime(){
        Date  time=new Date ();                    //创建 Date 对象
        //创建 SimpleDateFormat 对象，格式为年/月/日/小时/分钟/秒
```

```
        SimpleDateFormat simpleDateFormat=new SimpleDateFormat("yyyy MM dd
        HH mm ss");
        simpleDateFormat.format(time);          //格式化 Date 对象
        return time.toString();
    }
    public class MyBinder extends Binder{
    MyService getMyservice(){
        return MyService.this;
    }
    }
}
```

ServiceExample.java 代码如下：

```
package com.ServiceExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ServiceExample extends Activity {
    private MyService myService;              //声明 MyService 类型变量
    private TextView myTextView;              //声明 TextView 类型变量
    //ServiceConnection 监听 Service 状态的接口，在 Context.bindService 和
    context.unBindService()里用到
    private ServiceConnection myServiceConnection=new ServiceConnection(){
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            //TODO Auto-generated method stub
            myService=((MyService.MyBinder)service).getMyservice();
                                              //获取 MyBindder 对象
            myTextView.setText("time:"+myService.getSystemTime());
                                              //在 myTextView 上显示系统日期
        }
        @Override
        public void onServiceDisconnected(ComponentName name) {
            //TODO Auto-generated method stub

        }
    };
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);        //加载 main.xml 资源文件
        myTextView=(TextView) this.findViewById(R.id.showMessage);
                                              //获取资源文件中的 TextView 组件
        //获取资源文件中“启动 Service”的 Button 组件
        Button startServiceButton=(Button) this.findViewById(R.id.startServiceBut);
        //获取资源文件中“停止 Service”的 Button 组件
        Button stopServiceButton=(Button) this.findViewById(R.id.stopServiceBut);
        //获取资源文件中的 Button 组件
        Button bindServiceButton=(Button) this.findViewById(R.id.bindServiceBut);
        //获取资源文件中的 Button 组件
        Button unbindServiceButton=(Button) this.findViewById(R.id.unbind-
        ServiceBut);
        startServiceButton.setOnClickListener(new OnClickListener(){
                                              //监听 Button 组件的单击时间
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                Intent intent=new Intent();
                intent.setClass(ServiceExample.this, MyService.class);
```

```
                startService(intent);       //启动服务
            }
        });
        stopServiceButton.setOnClickListener(new OnClickListener(){
                                                //监听 Button 组件的单击时间
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                Intent intent=new Intent();
                intent.setClass(ServiceExample.this, MyService.class);
                stopService(intent);        //停止服务
            }
        });
        bindServiceButton.setOnClickListener(new OnClickListener(){
                                                //监听 Button 组件的单击时间
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                Intent intent=new Intent();
                intent.setClass(ServiceExample.this, MyService.class);
                bindService(intent,myServiceConnection,BIND AUTO CREATE);
                                                //连接应用服务
            }
        });
        unbindServiceButton.setOnClickListener(new OnClickListener(){
                                                //监听 Button 组件的单击时间
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                unbindService(myServiceConnection);
            }
        });
    }
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- <LinearLayout >：流式布局标签，关于布局详细内容将在第 3 章介绍-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout height="fill parent" android:background="#FFFFFFFF">
     <!--<TextVIew>：显示文本信息标签，关于 Android 的详细标签及组件介绍将在第 4 章
     介绍-->
    <TextView android:id="@+id/showMessage" android:layout width="wrap content"
    android:layout height="wrap content" android:text="测试 Service"
    android:textColor="#FF000000"/>
     <!--< Button >：按钮标签，关于 Android 的详细标签及组件介绍将在第 4 章介绍-->
    <Button android:id="@+id/startServiceBut" android:layout width="wrap content"
        android:layout_height="wrap_content" android:text="startService" />
    <Button android:id="@+id/stopServiceBut" android:layout width="wrap content"
        android:layout height="wrap content" android:text="stopService" />
    <Button android:id="@+id/bindServiceBut" android:layout width="wrap content"
        android:layout height="wrap content" android:text="bindService" />
    <Button android:id="@+id/unbindServiceBut" android:layout width=
    "wrap content"
        android:layout height="wrap content" android:text="unbindService" />
</LinearLayout>
```

需要在 AndroidManifest.xml 中声明 Service。AndroidManifest.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.ServiceExample"
     android:versionCode="1"
     android:versionName="1.0">
   <!--<application>: Android 应用标签。属性 android: icon: 表示应用的图标。
        android: label: 应用的名称。-->
   <application android:icon="@drawable/icon" android:label="@string/
   app name">
    <!-- <activity>: 窗体标签。属性 android: name: 窗体类名。
          Android: label: 窗体标题栏上显示的文字-->
      <activity android:name=".ServiceExample"
              android:label="@string/app name">
         <intent-filter>
            <!—设置该 Activity 为启动窗体-->
            <action android:name="android.intent.action.MAIN" />
             <!—设置该 Activity 在应用列表中显示-->
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
       <service android:name=".MyService" android:exported=
       "true"></service>
   </application>
</manifest>
```

2.3　Content Provider 介绍

在 Android 中，一个应用的数据库、文件等内容，都不允许其他应用直接访问，但有时候需要必要的数据共享。例如，如果不能对系统联系人列表进行添删改查，那么自己必须重新写一套方法，重头实现该功能。Android 不能让每个应用都是独立的孤岛，因此它为每个应用准备一个对外提供数据的方式，就是 Content Provider。Android 中提供了一系列内置的 Content Provider，如音乐、手机通讯联系人信息和图像，这些都在 android.provider 包下。在 AndroidManifest.xml 文件中添加权限许可，便可以在应用程序中访问这些 Content Provider。

访问 Content Provider 中的数据主要通过 ContentResolver 对象，在 ContentResolver 提供了可以对 Content Provider 中数据进行添、删、改、查等操作的方法。例如，查询 Content Provider 的方法有两个，分别是 ContentResolver 类中的 query()方法和 Activity 对象的 managedQuery()方法，这两个方法参数相同，返回的都是 Cursor 对象。不同的是 managedQuery()方法可以让 Activity 来管理 Cursor 的生命周期。

2.4　BroadcastReceiver 介绍

Broadcast 是一种在应用程序之间进行传输信息的机制。BroadcastReceiver 对发送出来

的 Broadcast 进行过滤并响应。广播 Intent 的发送是通过 Context.sentBroadcast()、Context.sentOrderedBroadcast()或者 Context.sendStickyBroadcast()方法来实现。通常一个广播 Intent 可以被订阅了此 Intent 的多个广播接受者所接收。

广播接收器只有一个回调方法，即 void onReceive(Context curContext, Intent broadcastMsg)方法，当广播消息抵达接收器时，系统将调用 onReceive()方法并且把包括消息的 Intent 对象传递给它。广播接收器只有在执行这个方法的时候才处于活跃状态，当该方法执行完毕后，广播接收器处于失活状态。

下面的例子实现监听电池电量信息。

BroadcastExample .java 文件代码如下：

```
package com.BroadcastExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class BroadcastExample extends Activity {
    private TextView batteryView;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);          //加载布局资源文件
        batteryView = (TextView) this.findViewById(R.id.batteryView);
                                                //获取资源文件中的 TextView
    }

    @Override
    protected void onPause() {
        //TODO Auto-generated method stub
        super.onPause();
        unregisterReceiver(batteryInfoReceiver);   //注销广播监听器
    }

    @Override
    protected void onResume() {
        //TODO Auto-generated method stub
        super.onResume();
        //注册电池电量广播监听器
        registerReceiver(batteryInfoReceiver, new IntentFilter(
                Intent.ACTION_BATTERY_CHANGED));
    }

    private BroadcastReceiver batteryInfoReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            //TODO Auto-generated method stub
            if (Intent.ACTION_BATTERY_CHANGED.equals(intent.getAction())) {
                int level = intent.getIntExtra("level", 0);
                int scale = intent.getIntExtra("scale", 100);
                batteryView.setText("Battery Level:"
                        + String.valueOf(level * 100 / scale) + "%");
                                                   //显示电量信息
            }
        }

    };
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFFFFFFF">
    <TextView android:id="@+id/batteryView" android:layout_width="fill_
    parent"
        android:layout_height="wrap_content" android:textColor="#FF000000" />
</LinearLayout>
```

运行结果如图 2.10 所示。

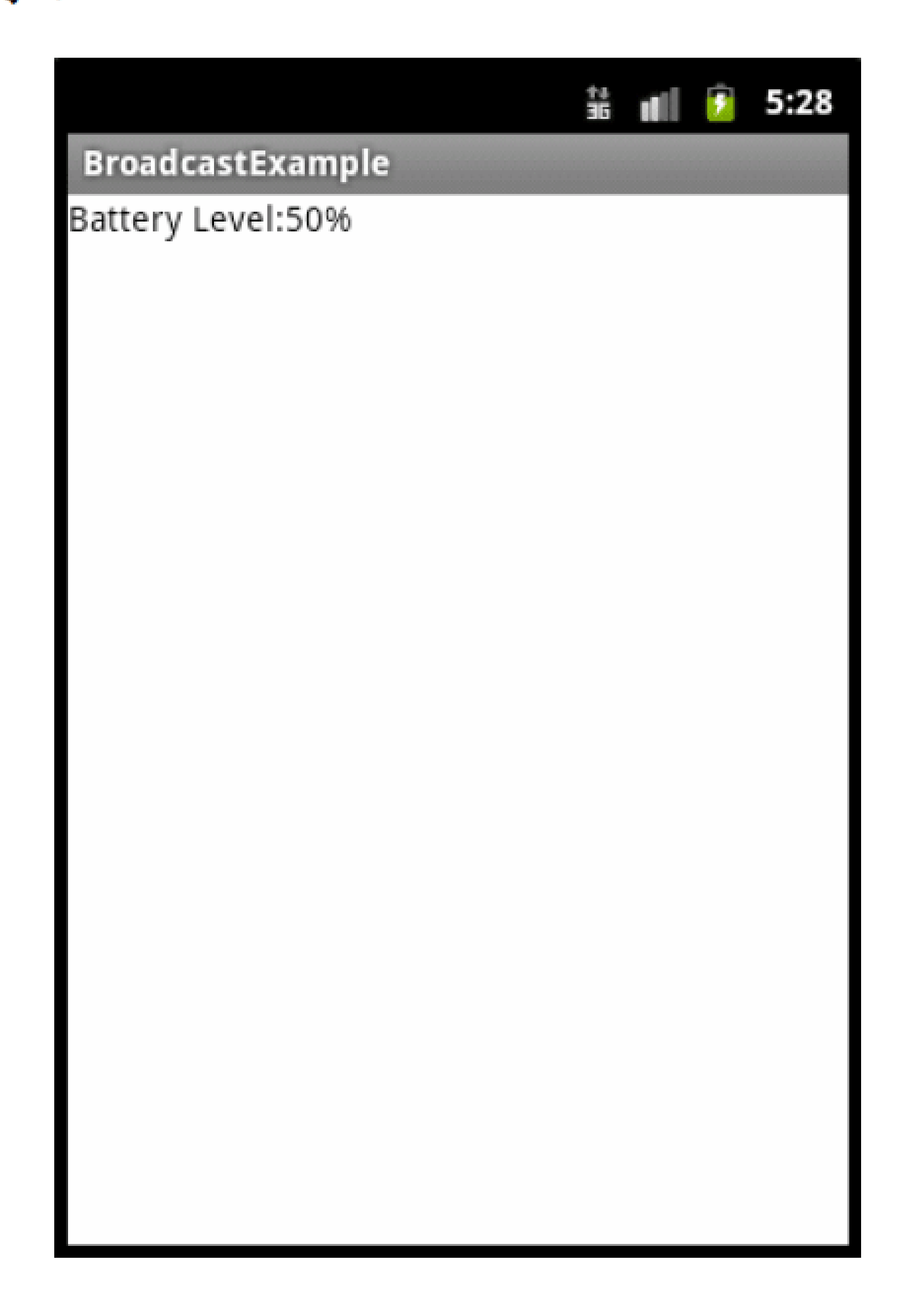


图 2.10　BroadcastReceiver 示例

第 2 篇　Android 应用开发实例

第 3 章　Android UI 布局

通过前面的学习，我们已经知道了搭建 Android 这座房子所需要的工具，以及未来这座房子可以提供给我们哪些服务。接下来就需要我们亲手来设计房子，你的房子格局如何设计、家具如何摆放，这就是 Android UI 布局和组件所需要做的事情。

3.1　使用 XML 资源创建布局

Android 用户界面的布局可以通过两种方式来实现，第一种方式是通过 XML 资源文件创建布局，这种方式类似于我们通过 HTML 标签设计页面，实现起来比较简单。第二种方式是通过代码实现用户界面布局，这种方式类似于 Java 中的 Swing 组件布局，实现起来相对复杂。在这一节中我们了解一下如何通过 XML 资源文件创建布局。

通过 XML 文件我们可以定义窗口组件之间的关系，以及窗口组件和容器之间的关系。Android 系统把布局文件作为一种资源，存储在项目的 res/layout 目录下。

每个布局文件中包括了一个树状的元素集合，代表了窗口组件和容器之间的关系，每个 XML 元素都有一些属性，用于描述窗口组件的特性，如颜色、背景等。下面我们来建立一个 XML 文件，认识一下如何通过 XML 文件创建布局。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- < LinearLayout >: 流式布局标签-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent"  android:background="#ffF4F4F4" >
    <!-- <TextView>: 用来显示提示信息的标签 -->
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="师徒四人去取经"
        android:gravity="center" android:textSize="20px" android:text-
        Color="#ffff0000" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="唐僧"
        android:textSize="18px" android:textColor="#ff0000ff" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="悟空"
        android:textSize="18px" android:textColor="#ff0000ff" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="八戒"
        android:textSize="18px" android:textColor="#ff0000ff" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="沙僧"
        android:textSize="18px" android:textColor="#ff0000ff" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:autoLink="all"
```

```
        android:text="精彩片段请访问 http://www.xitian.com" android:text-
        Size="18px"
        android:textColor="#ff00ff" />
</LinearLayout>
```

代码说明如下。

- <LinearLayout>：流式布局元素标签。
- xmlns:android：LinearLayout 元素的属性，指定 XML 的命名空间，告诉 Android 的开发工具使用该命名空间中的元素和属性。所有的 Android 设计文件必须引入这个命名空间。
- android:orientation：该属性用于指定流式布局的方向。流式布局有两种布局方向，分别为“vertical”表示垂直布局，“horizontal”表示水平布局。
- android:layout_width：该属性用于指定元素的水平宽度，有 3 种取值，分别为“fill_parent”表示当前元素的宽度由父元素的宽度决定，即会填充父元素，该值在 Android 2.2 开始由“match_parent” 所取代，他们的定义本质都是一样的，值均为-1，只是换了个别名。“wrap_content”表示当前元素的宽度由元素本身的内容决定，即根据内容的不同自动调节元素的宽度。例如，android:layout_width=“100px”或者 android:layout_width=“100dip”。
- android:layout_height：该属性用于指定元素的垂直高度，取值和 android:layout_width 取值相同，这里不再赘述。
- android:background：该属性用于设置元素的背景颜色，其中颜色值前两位表示透明度，“00”为透明，“ff”为不透明。
- <TextView>：文本显示元素标签，所有用于显示的信息都可以通过该元素实现。
- android:text：该属性指定元素上要显示的信息。
- android:textSize：该属性指定元素上文字的大小。
- android:textColor：该属性指定元素上文字的颜色。
- android:autoLink：该属性用于控制当显示信息中出现如 url、电话号码和 Email 等是否转化成连接形式，常用的取值有“none”为不匹配任何模式，默认是该选项。“web”为匹配 url 格式，“Email”为匹配电邮地址格式，“phone”为匹配电话号码格式，“all”为匹配全部格式。



图 3.1　XML 方式创建布局

运行结果如图 3.1 所示。

3.2　View 及 ViewGroup 简介

通过前面的学习，我们知道 Activity 是 Android 应用程序基本功能单元，Android 的每一个显示窗口都是一个 Activity。但是 Activity 本身并不能显示出来，我们要显示界面信息，需要在 Activity 中通过调用方法 setContentView(View view) 来实现，也就是说真正显示出来的是 View。

View 是 Android 组件的一个基类，一个 View 就是屏幕上的一个矩形区域，主要负责绘制元素和事件处理。图 3.2 为 View 的类层级图，我们可以看到 View 提供了很多用于界面显示的子类，例如按钮类 Button、复选框类 CheckBox 等，我们往往用 View 的这些子类去绘制界面元素。

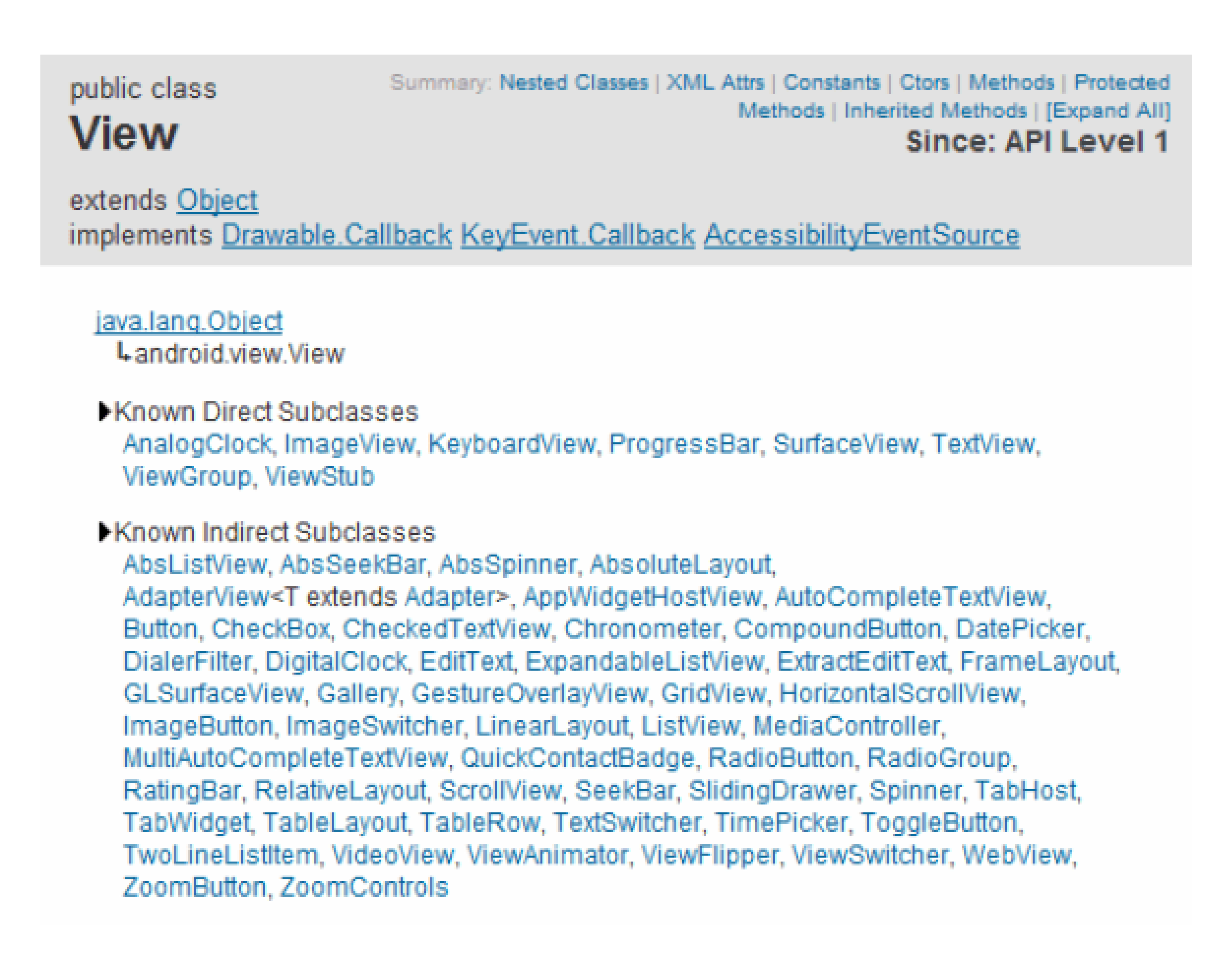


图 3.2　View 的类层级图

ViewGroup 是 View 的子类，它并不会直接显示出来，而是负责管理和容纳下一层的 View 和 ViewGroup，即它是容纳其他元素的容器。在 ViewGroup 中定义了嵌套类 ViewGroup.LayoutParams，这个类定义了显示对象时的属性，例如，显示对象的宽度、高度、位置等，View 通过 LayoutParams 告诉父窗体它的摆放位置。ViewGroup 是一个抽象类，如图 3.3 所示，所以真正充当容器角色的是它的子类们，例如 AbsoluteLayout 绝对布局、FrameLayout 帧布局、LinearLayout 流式布局、RelativeLayout 相对布局、TableLayout 表格布局等这些布局管理器。

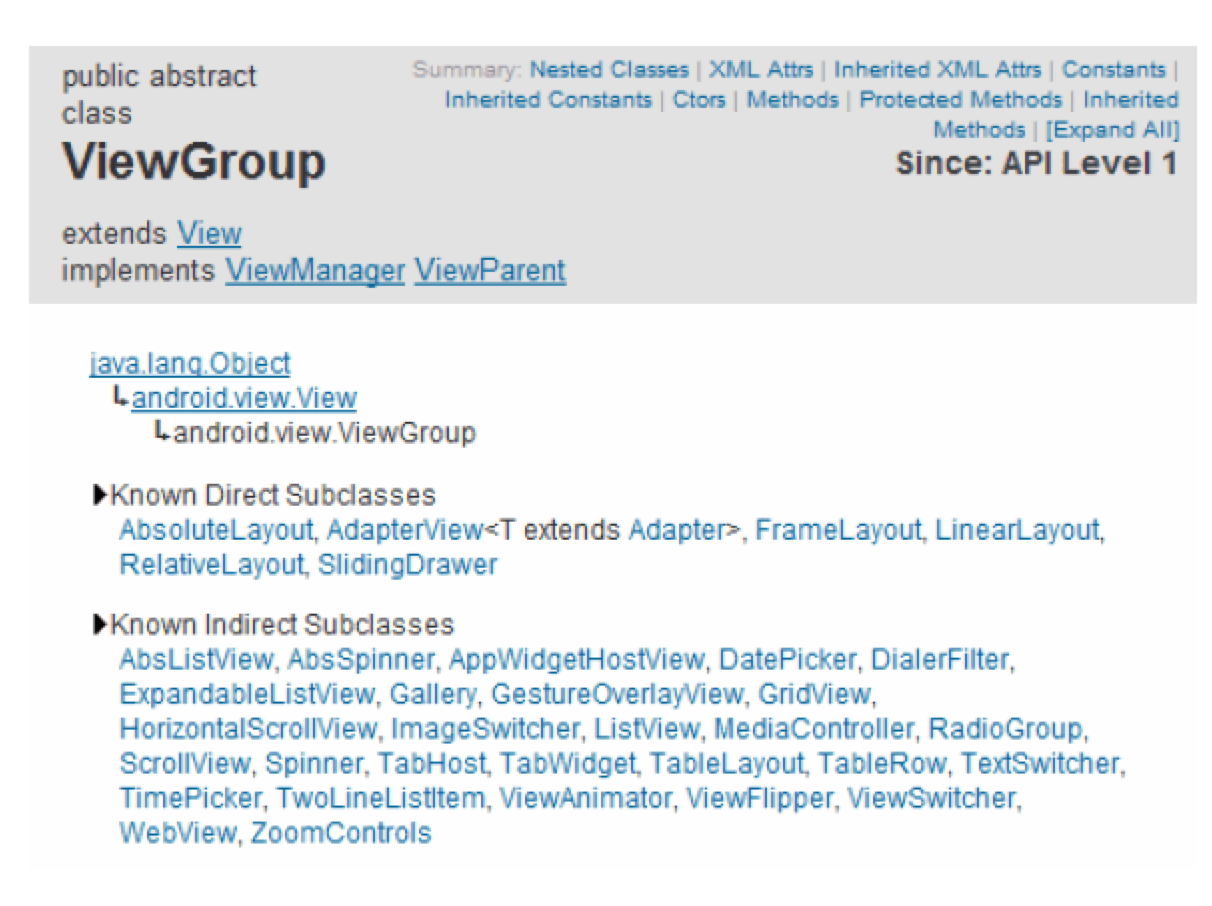


图 3.3　ViewGroup 的类层级图

3.3　普通布局对象

通过上一节的学习，我们对 Android 的 UI 组件层次关系有所了解，这节主要介绍常用

的布局管理器对象。布局管理器对象可以嵌套定义，即在一个布局管理器中可以包含其他的布局管理器。

3.3.1　FrameLayout 介绍及案例

FrameLayout 称为帧式布局或者框架布局，是最简单的一种布局对象，在它上面可以添加多个组件，所有组件都被固定在界面的左上角，叠加显示，后一个组件会在前一个组件之上显示，也就是说后一个组件会全部或部分的覆盖前一个组件。通过下面的例子了解一下 FrameLayout，在窗体上添加两个 TextView 组件。代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!--< FrameLayout >：帧式布局元素标签-->
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:background="#ffF4F4F4">
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="我在底层呢"
        android:textSize="20px" android:textColor="#ffff0000" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="我在顶层"
        android:textSize="18px" android:textColor="#ff0000ff" />
</FrameLayout>
```

代码说明如下。

<FrameLayout >：帧式布局元素标签，其他属性含义参考 3.1.1 节内容

运行结果如图 3.4 所示。



图 3.4　FrameLayout 布局

3.3.2　LinearLayout 介绍及案例

LinearLayout 称为流式布局或者线性布局，在流式布局管理器中，可以放置多个组件，所有的组件都可以按某个方向（水平或者垂直）对齐摆放，摆放方向通过<LinearLayout>元素属性 android:orientation 来设置。垂直摆放时，所有的组件都在一列显示，且每行将只有一个组件；水平摆放时，所有的组件都在一行显示。图 3.5 就是一个典型的流式布局，所有的组件都是垂直摆放。下面的例子展示了流式布局中水平摆放组件的效果。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:background="#ffF4F4F4"
    android:layout_width="fill_parent" android:layout_height="fill_parent">
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="我在前面,"
        android:textSize="20px" android:textColor="#ffff0000" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="我在后面，我们在
        一行"
        android:textSize="18px" android:textColor="#ff0000ff" />
</LinearLayout>
```

运行结果如图 3.5 所示。

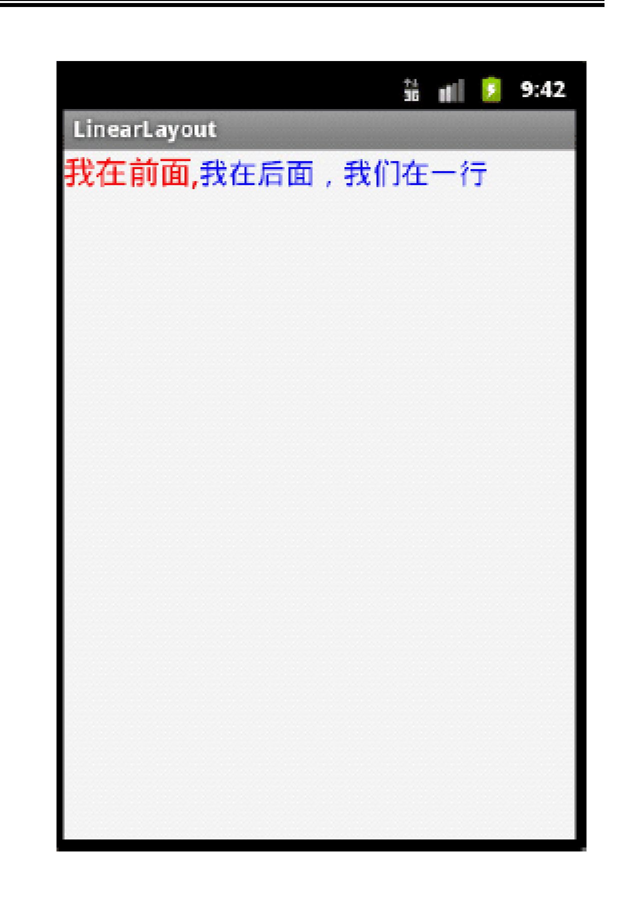


图 3.5　LinearLayout 布局

3.3.3　AbsoluteLayout 介绍及案例

AbsoluteLayout 称为绝对布局或者坐标布局。绝对布局是一种很灵活的布局方式，所有放置在它上面的组件可以通过 X/Y 坐标来定位，该坐标为相对于屏幕左上角（0,0）的值。但这种布局方式并不推荐，除非你有很好的理由去使用它，因为不同规格的设备尺寸不同，它有可能不能很好地显示统一的效果。通过下面地例子我们了解一下绝对布局对象。

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent" android:layout_height="fill_
        parent"
        android:background="#ffF4F4F4">
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Good Good study! "
        android:textSize="20px" android:textColor="#ffff0000"
        android:layout_x="20px" android:layout_y="50px" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="Day Day up! "
        android:textSize="18px" android:textColor="#ff0000ff"
        android:layout_x="40px" android:layout_y="90px"/>
</AbsoluteLayout>
```

代码说明如下。

- <AbsoluteLayout>：绝对布局元素标签。
- android:layout_x：该属性指定 TextView 组件出现在屏幕上的 X 坐标值。
- android:layout_y：该属性指定 TextView 组件出现在屏幕上的 Y 坐标值。

注意：如果不指定 android:layout_x 和 android:layout_y，组件默认出现在屏幕左上角，即（0,0）位置。

运行结果如图 3.6 所示。

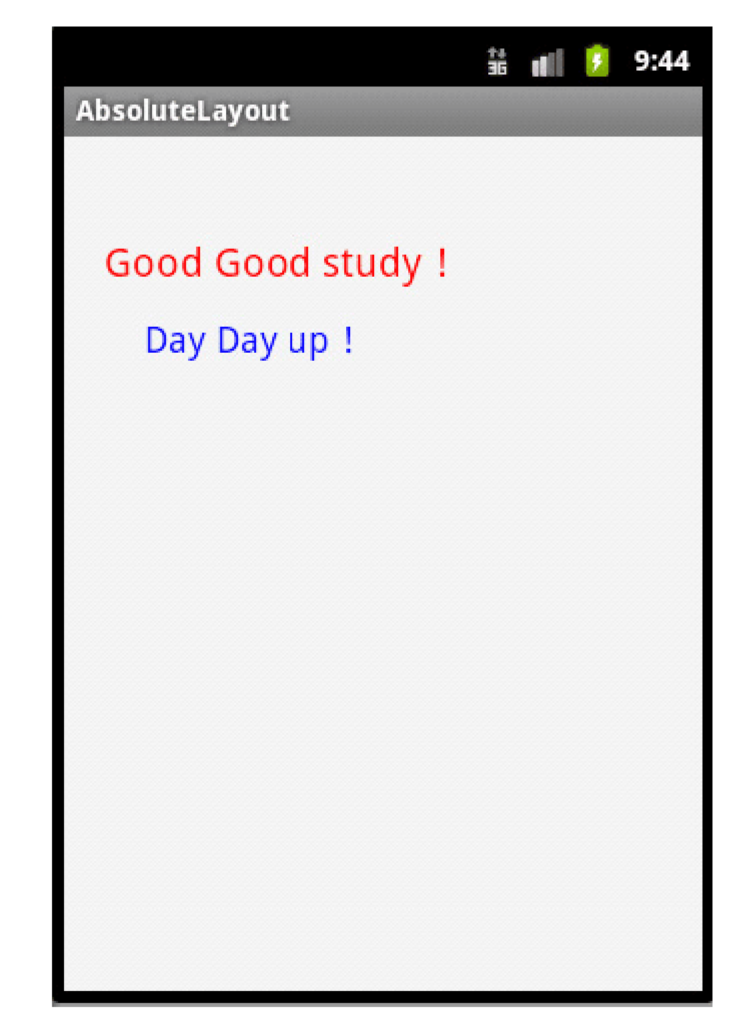


图 3.6　AbsoluteLayout 布局

3.3.4　RelativeLayout 介绍及案例

RelativeLayout 称为相对布局，放置在相对布局上的组件可以设置它相对于子元素或者父元素的位置，通过指定 ID 来指定相对于哪个子元素去定位。相对布局当界面组件较多时，实现起来相对复杂。通过下面的例子，我们了解一下相对布局的使用。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
```

```
    android:background="#ffF4F4F4" android:padding="10px">
    <TextView android:id="@+id/tips" android:layout width="fill parent"
        android:layout height="wrap content" android:text="口令: "
        android:textSize="16px" android:textColor="#ffff0000" />
    <EditText android:id="@+id/tipsEdit" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:hint="请输入口令"
        android:textSize="16px" android:textColor="#ff0000ff"
        android:layout below="@+id/tips" />
    <Button android:id="@+id/cancelButton" android:layout width="wrap
    content"
        android:layout height="wrap content" android:hint="取    消"
        android:layout_alignParentRight="true" android:textSize="16px"
        android:layout_below="@+id/tipsEdit" />
    <Button android:id="@+id/okButton" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:hint="确    定"
        android:textSize="16px" android:layout alignTop="@+id/cancelButton"
        android:layout marginLeft="150px" />
</RelativeLayout>
```

代码说明如下。

- <RelativeLayout>：相对布局元素标签。
- android:padding：定义该组件的边内补白，与 CSS 中的 padding 意义相同，即定义了组件边框与组件里面内容的距离。
- android:id：组件的唯一标识，在 XML 中一般可以通过@+语法去创建 ID，例如 android：id=“@+id/myId”，我们可以在程序代码中通过 findViewById(R.id.myId) 获取该组件。
- android:layout_below：该属性定义了当前组件在指定 ID 组件的下面，在这里标识 ID 为 tipsEdit 的组件在 ID 为 tips 组件的下面。
- android:layout_alignParentRight：当前组件的右端是否与父窗体的右端对齐，取值为 true 或 false。
- android:layout_alignTop：当前组件与 android:layout_alignTop 属性指定 ID 的组件上端对齐，这里表示 ID 为 okButton 的按钮，与 ID 为 cancelButton 的按钮上端对齐。
- android:layout_marginLeft：设置组件的左边距，这里表示 ID 为 okButton 的组件距离窗体左边框的距离是 150px。

运行结果如图 3.7 所示。



图 3.7　RelativeLayout 布局

3.3.5　TableLayout 介绍及案例

TableLayout 称为表格布局，和 HTML 中的<table>标签类似，将子元素分配到行或列中，但 TableLayout 没有边框，一个 TableLayout 由多个 TableRow 组成，每个 TableRow 中可以有 0 或者多个单元格。每个单元格都是一个 View，单元格的索引从 0 开始。可以通

过 android:layout_span 实现合并单元格，但不支持跨行合并。也可以通过 android:layout_column 指定组件要显示在哪个单元格位置，从而跳过一些单元格。下面看一个表格布局的例子，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:background="#ffF4F4F4" android:stretchColumns="1">
    <TableRow>
        <TextView android:text="TableLayout Example" android:padding="3px"
            android:textColor="#FF909090" android:layout_span="3"
            android:gravity="center" android:background="#FEF284"/>
    </TableRow>
    <!-- 间隔线 -->
    <View android:layout_height="2dip" android:background="#FF909090" />
    <TableRow>
        <TextView android:text="New" android:padding="3px"
            android:textColor="#FF909090" android:background="#FEF284"/>
        <TextView android:text="Alt+Shift+N" android:gravity="right"
            android:padding="3px" android:textColor="#FF909090"
            android:background="#A6BFF2"/>
    </TableRow>

    <TableRow>
        <TextView android:text="Open File..." android:padding="3px"
            android:textColor="#FF909090" android:background="#FEF284"/>
    </TableRow>
    <View android:layout_height="2dip" android:background="#FF909090" />
    <TableRow android:layout_width="match_parent">
        <TextView android:text="Close" android:padding="3px"
            android:textColor="#FF909090"  android:background="#FEF284"/>
        <TextView android:text="Ctrl+W" android:gravity="right"
            android:padding="3px" android:textColor="#FF909090"
            android:background="#A6BFF2"/>
    </TableRow>

    <TableRow>
        <TextView android:text="Close" android:padding="3px"
            android:textColor="#FF909090"
            android:background="#FEF284"/>
        <TextView android:text="Ctrl+Shift+W" android:gravity="right"
            android:padding="3px" android:textColor="#FF909090"
            android:background="#A6BFF2"/>
    </TableRow>
    <View android:layout_height="2dip" android:background="#FF909090" />

    <TableRow>
        <TextView android:text="Exit" android:padding="3px"
            android:textColor="#FF909090"  android:background="#FEF284"/>
    </TableRow>
    <TableRow>
        <TextView android:text="END" android:padding="3px"
            android:textColor="#FF909090" android:layout_column="1"
            android:gravity="center"  android:background="#FEF284"/>
    </TableRow>
</TableLayout>
```

代码说明如下。

- <TableLayout>：表格布局元素标签。

- android:stretchColumns：该属性用来设置某列宽度为自动拉伸，列的索引从 0 开始，这里设置的值为 1，表示第二列的宽度自动拉伸，即除了第一列每行剩余的空间都会分配给第二列。
- <TableRow>：行元素标签。
- android:layout_span：该属性用来设置合并单元格，这里表示合并两个单元格。
- android:gravity：设置此组件中的内容在组件中的位置，可选的值有 top、bottom、left、right、center_vertical、fill_vertical、center_horizontal、fill_horizontal、center、fill 和 clip_vertical，可以多选，用“|”分开。
- android:layout_column：该属性设置组件显示在哪个单元格中。这里设置的值为 1，表示该组件显示在第二个单元格中。

运行结果如图 3.8 所示。

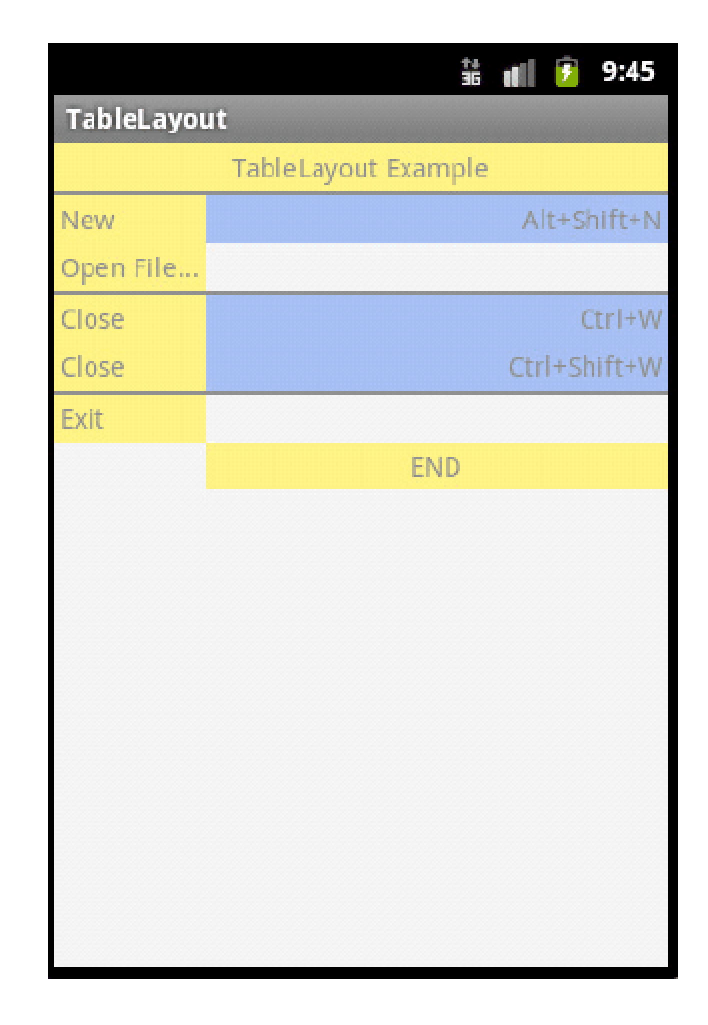


图 3.8　TableLayout 布局

3.4　使用 TabActivity 和 TabHost 组织视图

TabHost 是一个装载 Tab 的容器，每个 Tab 项中可以加载一个布局，可以通过 TabActivity 的 getTabHost()方法获取 TabHost 对象，TabHost 中提供了添加 Tab 页及修改 Tab 页的方法。Tab 实现的效果如图 3.9 所示。下面我们看一个示例，Book2Tab1.java 代码如下：

```
package com.AndroidBook;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class Book2Tab1 extends TabActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setTitle("TabActivity 和 TabHost");          //设置窗体的标题
        TabHost tabHost = this.getTabHost();
                                   //从 TabActivity 上获取 TabHost 容器
        LayoutInflater.from(this).inflate(R.layout.book2tab1,tabHost.
        getTabContentView(), true);
        tabHost.addTab(tabHost.newTabSpec("tab1").setIndicator("红色",
                getResources().getDrawable(R.drawable.homemenu)).
                setContent(R.id.view1));
        tabHost.addTab(tabHost.newTabSpec("tab2").setIndicator("绿色").
        setContent(R.id.view2));
        tabHost.addTab(tabHost.newTabSpec("tab3").setIndicator("蓝色").
        setContent(R.id.view3));
    }
}
```

该示例中 LayoutInflater 类及方法说明如下。

- LayoutInflater：是一个抽象类，用来实例化 XML 布局文件生成对应的 View。

- public static LayoutInflater from(Context context)：该方法由 LayoutInflater 类提供。该方法是从当前给定的 Context 获取 LayoutInflater 对象。参数是 Context 类型，Context 是一个抽象类，是一个提供了关于应用环境全局信息的接口，由 Android 系统调用执行，通过它可以访问应用的资源，同时启动应用级的操作。代码中的 LayoutInflater.from(this)是获取当前 TabActivity 的 LayoutInflater 对象。
- public View inflate(int resource, ViewGroup root, boolean attachToRoot)：获取指定 XML 资源文件的 View。第 1 个参数为要加载的 XML 布局资源 ID，如果第 3 个参数为 true，则第 2 个参数表示将指定的 XML 布局资源挂靠在 root 上。

TabHost 类和方法说明如下。

- TabHost：装载 Tab 的容器。
- public void addTab (TabHost.TabSpec tabSpec)：该方法用来添加 Tab 标签。参数为内嵌类。

TabHost.TabSpec 类和方法说明如下。

- TabHost.TabSpec：通过该内嵌类可以为 Tab 添加标签、内容、图标等。
- public TabHost.TabSpec newTabSpec(String tag)：该方法用于在 TabHost 上添加一个新的 Tab 页。参数为 Tab 标签的名字。返回值为 TabHost.TabSpec。
- public TabHost.TabSpec setIndicator(CharSequence label)：为 Tab 按钮上添加文字。参数为 Tab 上要显示的文字。返回值为 TabHost.TabSpec 类型。程序中 setIndicator ("绿色")即调用该方法来设置第二个标签上的显示文字。
- public TabHost.TabSpec setIndicator(CharSequence label, Drawable icon)：为 Tab 按钮上添加文字和图标信息，第 1 个参数是 Tab 上要显示的文字信息，第 2 个参数为 Tab 上要显示的图标信息。返回值为 TabHost.TabSpec 类型。程序中 setIndicator ("红色",getResources().getDrawable(R.drawable.homemenu))即调用该方法为第一个 Tab 页设置显示文字的同时设置图标。
- public TabHost.TabSpec setContent(int viewId)：Tab 显示的信息为指定 ID 的 View 的内容。例如，该程序中指定了 ID 为 view1 的 TextView 为第一个 Tab 的内容。
- public TabHost.TabSpec setContent (Intent intent)：指定一个 Intent 作为 Tab 的内容。在后面的示例中我们会用到该方法。

res/layout/ book2tab1.xml 布局资源文件代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">
    <!-- tab1 的内容显示区域 -->
    <TextView android:id="@+id/view1" android:background="#ffff0000"
        android:layout_width="match_parent" android:layout_height=
        "match_parent"
        android:text="红色区域" />
    <!-- tab2 的内容显示区域 -->
    <TextView android:id="@+id/view2" android:background="#ff00ff00"
        android:layout_width="match_parent" android:layout_height=
        "match_parent"
        android:text="绿色区域" />
    <!-- tab3 的内容显示区域 -->
    <TextView android:id="@+id/view3" android:background="#ff0000ff"
```

```
        android:layout width="match parent" android:layout_height=
        "match parent"
        android:text="蓝色区域" />
</FrameLayout>
```

运行结果如图 3.9 所示。

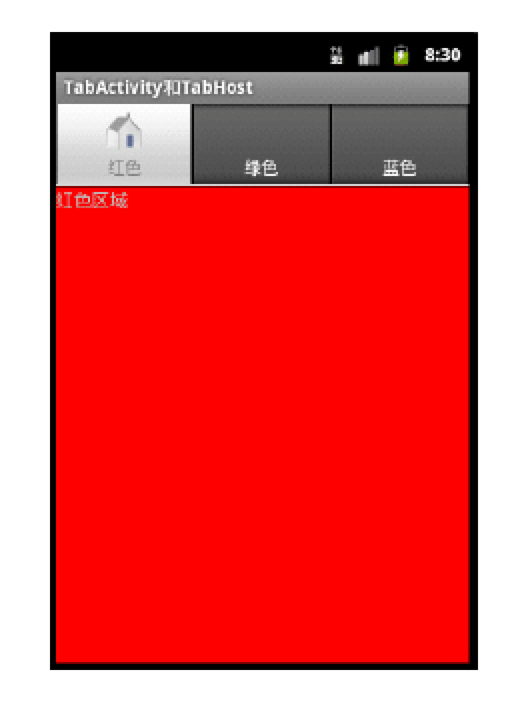


图 3.9　TabActivity 和 TabHost 示例

在图 3.9 中，单击不同 Tab 显示的内容，是通过加载 XML 资源文件中的 TextView 组件来实现的，而在实际项目中单击不同的 Tab 往往需要加载不同的窗体信息。在 TabHost.TabSpec 类中，提供了一个 setContent(Intent intent)方法，可以将 Intent 作为 Tab 的内容。对上面的 Java 代码进行修改，修改后如下所示。

```
package com.AndroidBook;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class Book2Tab2 extends TabActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setTitle("TabActivity 和 TabHost");        //设置窗体的标题
        TabHost tabHost = this.getTabHost();
                                    //从 TabActivity 上获取 TabHost 容器
        tabHost.addTab(tabHost.newTabSpec("tab1").setIndicator("相对布局
        示例",
                getResources().getDrawable(R.drawable.homemenu)).setContent
                (new Intent(this,Book2Relativelayout.class)));
        tabHost.addTab(tabHost.newTabSpec("tab2").setIndicator("表格布局
        示例").setContent(new Intent(this,Book2TableLayout.class)));
    }
}
```

代码说明如下。

这里 setContent()方法的参数是 Intent 对象，单击“相对布局示例”的 Tab，启动 3.3.4 节 RelativeLayout 的 Activity，如图 3.10 所示。单击“表格布局示例”的 Tab，启动 3.3.5 节 TableLayout 的 Activity，运行效果如图 3.11 所示。



图 3.10　TabActivity 和 TabHost 示例



图 3.11　TabActivity 和 TabHost 示例

3.5　布局的嵌套使用

在实际开发过程中，往往涉及的界面比较复杂，不是某一个单一布局可以很好实现的，这时候就需要布局的嵌套使用，以及同时使用多种布局来实现一个复杂的界面。下面我们来看一下示例 1，如图 3.12 所示，为 LinearLayout 布局的嵌套效果。

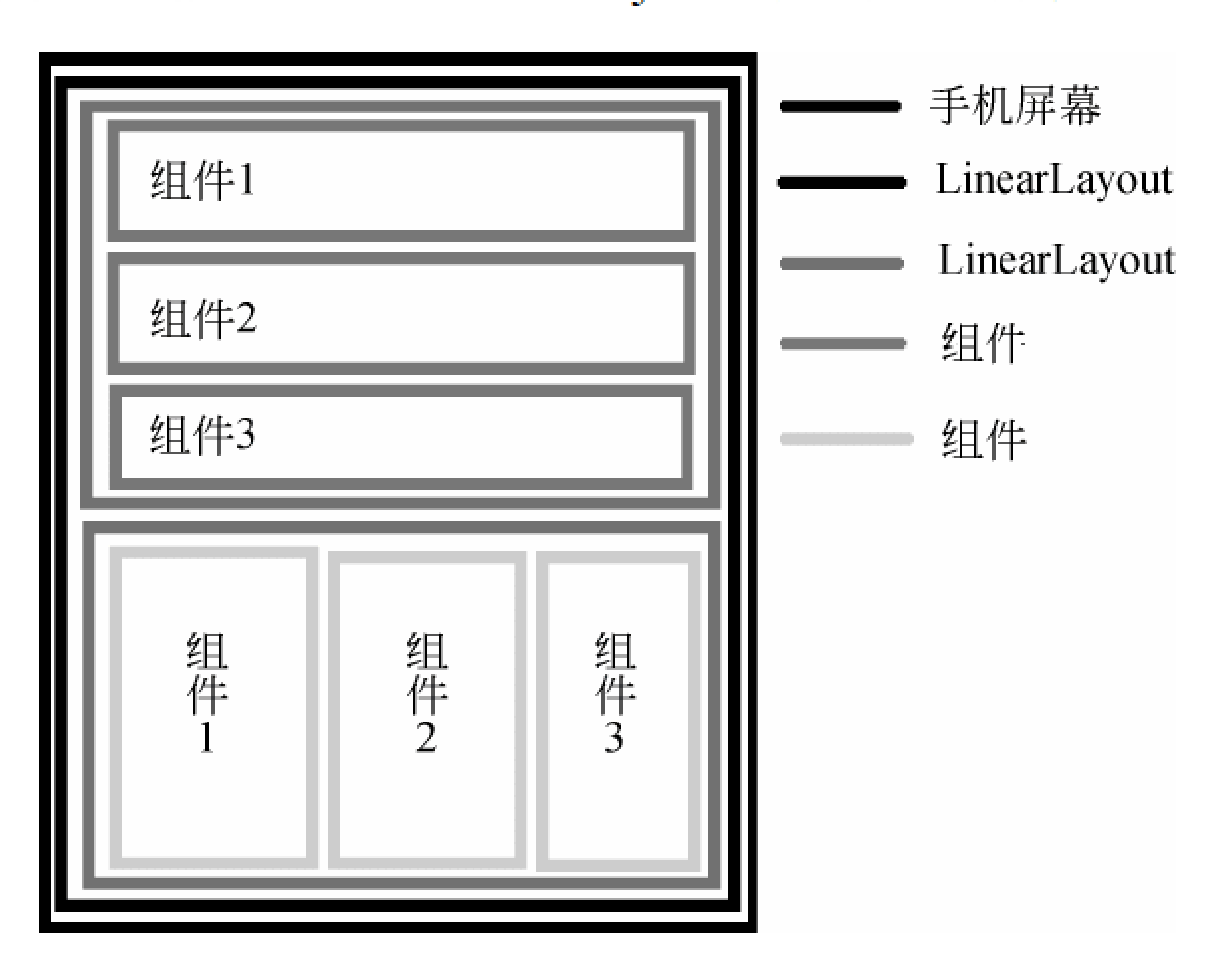


图 3.12　LinearLayout 布局的嵌套使用

实现上图效果的 XML 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 外层 LinearLayout 布局，垂直方向 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout height="fill parent" android:background="#FFF0F0F0">
    <!--流式布局，垂直方向 -->
    <LinearLayout android:layout width="fill parent"
        android:layout height="fill parent" android:orientation="vertical"
        android:layout_weight="1">
        <TextView android:layout_width="wrap_content"
            android:layout_height="fill_parent" android:text="组件 1"
            android:textSize="15px" android:textColor="#FF000000"
            android:layout_weight="1" android:gravity="center_vertical" />
        <TextView android:layout width="wrap content"
            android:layout_height="fill_parent" android:text="组件 2"
            android:textSize="15px" android:textColor="#FF000000"
            android:layout_weight="1" android:gravity="center_vertical" />
        <TextView android:layout width="wrap content"
            android:layout height="fill parent" android:text="组件 3"
            android:textSize="15px" android:textColor="#FF000000"
            android:layout weight="1" android:gravity="center vertical" />
    </LinearLayout>
    <!--流式布局，水平方向 -->
    <LinearLayout android:layout width="fill parent"
        android:layout height="fill parent" android:orientation="horizontal"
        android:layout_weight="1">
```

```
        <TextView android:layout_width="wrap_content"
            android:layout_height="fill_parent" android:text="组件 1"
            android:textSize="15px" android:textColor="#FF000000"
            android:layout_weight="1" android:background="#FFFF0000" />
        <TextView android:layout_width="wrap_content"
            android:layout_height="fill_parent" android:text="组件 2"
            android:textSize="15px" android:textColor="#FF000000"
            android:layout_weight="1" android:background="#FF00FF00" />
        <TextView android:layout_width="wrap_content"
            android:layout_height="fill_parent" android:text="组件 3"
            android:textSize="15px" android:textColor="#FF000000"
            android:layout_weight="1" android:background="#FF0000FF" />
    </LinearLayout>
</LinearLayout>
```

运行效果如图 3.13 所示。

下面的示例 2 为 LinearLayout 和 TableLayout 嵌套布局实现效果，如图 3.14 所示为效果实现布局参考图。

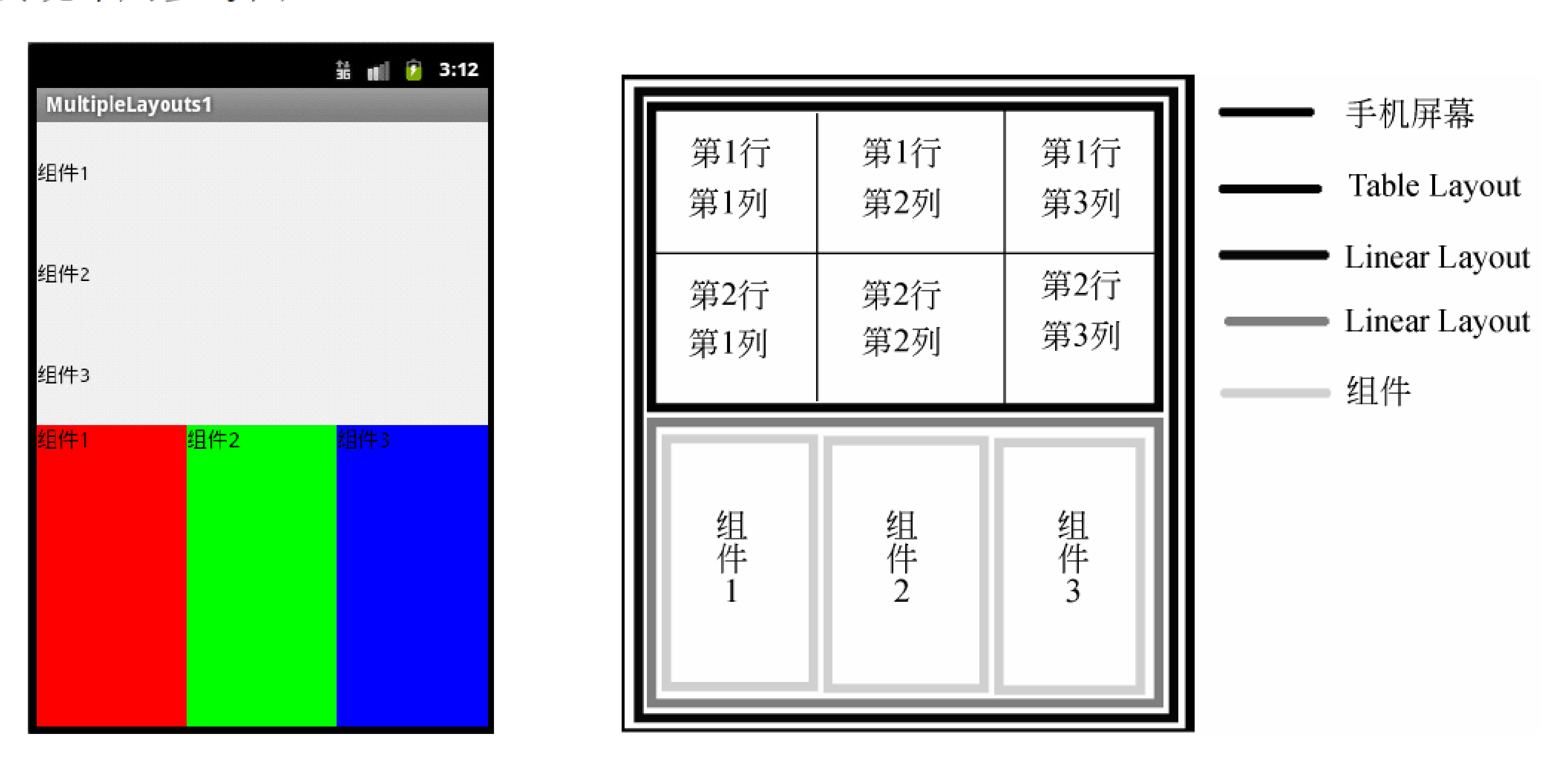


图 3.13　LinearLayout 布局的嵌套使用　图 3.14　LinearLayout 布局和 TableLayout 布局的嵌套使用

实现图 3.14 所示效果的 XML 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 外层 LinearLayout 布局，垂直方向 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFF0F0F0">
    <!-- 表格布局 -->
    <TableLayout android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="1">
        <TableRow android:layout_weight="1">
            <TextView android:layout_width="wrap_content"
                android:layout_height="fill_parent" android:text="第 1 行，
                第 1 列"
                android:textSize="15px" android:textColor="#FF000000"
                android:layout_weight="1" android:background="#FFFF0000" />
```

```
            <TextView android:layout width="wrap content"
                android:layout_height="fill_parent" android:text="第 1 行，
                第 2 列"
                android:textSize="15px" android:textColor="#FF000000"
                android:layout weight="1" android:background="#FF00FF00" />
            <TextView android:layout width="wrap content"
                android:layout_height="fill_parent" android:text="第 1 行，
                第 3 列"
                android:textSize="15px" android:textColor="#FF000000"
                android:layout weight="1" android:background="#FF0000FF" />
        </TableRow>
        <TableRow android:layout weight="1">
            <TextView android:layout width="wrap content"
                android:layout_height="fill_parent" android:text="第 2 行，
                第 1 列"
                android:textSize="15px" android:textColor="#FF000000"
                android:layout weight="1" android:background="#FFFFFF00" />
            <TextView android:layout width="wrap content"
                android:layout_height="fill_parent" android:text="第 2 行，
                第 2 列"
                android:textSize="15px" android:textColor="#FF000000"
                android:layout weight="1" android:background="#FF00FFFF" />
            <TextView android:layout width="wrap content"
                android:layout_height="fill_parent" android:text="第 2 行，
                第 3 列"
                android:textSize="15px" android:textColor="#FF000000"
                android:layout weight="1" android:background="#FFFF00FF" />
        </TableRow>
    </TableLayout>
    <!-- 流式布局 -->
    <LinearLayout android:layout width="fill parent"
        android:layout height="fill parent" android:orientation="horizontal"
        android:layout weight="1" android:layout marginTop="5px">
        <TextView android:layout width="wrap content"
            android:layout_height="fill_parent" android:text="组件 1"
            android:textSize="15px" android:textColor="#FF000000"
            android:layout weight="1" android:background="#FFFF0000" />
        <TextView android:layout width="wrap content"
            android:layout height="fill parent" android:text="组件 2"
            android:textSize="15px" android:textColor="#FF000000"
            android:layout weight="1" android:background="#FF00FF00" />
        <TextView android:layout width="wrap content"
            android:layout height="fill parent" android:text="组件 3"
            android:textSize="15px" android:textColor="#FF000000"
            android:layout weight="1" android:background="#FF0000FF" />
    </LinearLayout>
</LinearLayout>
```

运行效果如图 3.15 所示。

看了上面两个示例，我们对嵌套布局的使用已有所了解，下面我们以手机版 QQ 登录页面为例，看看在实际应用中多种布局嵌套的使用。图 3.16 为 QQ 登录布局设计参考图。

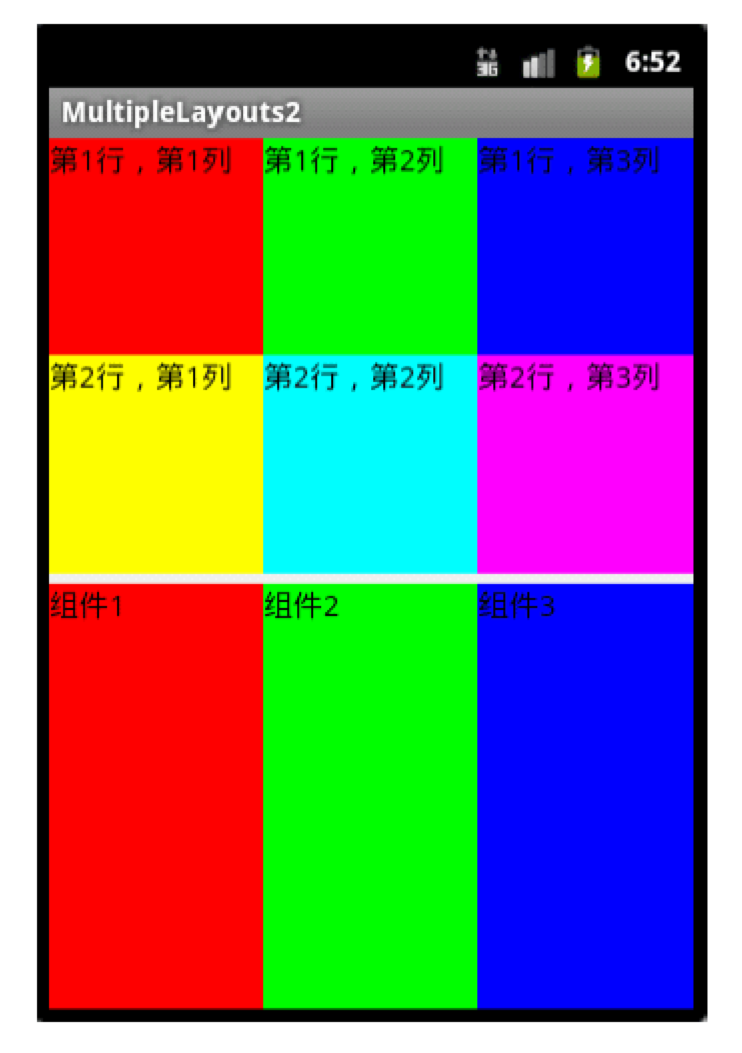


图 3.15　LinearLayout 布局和 TableLayout 布局的嵌套使用

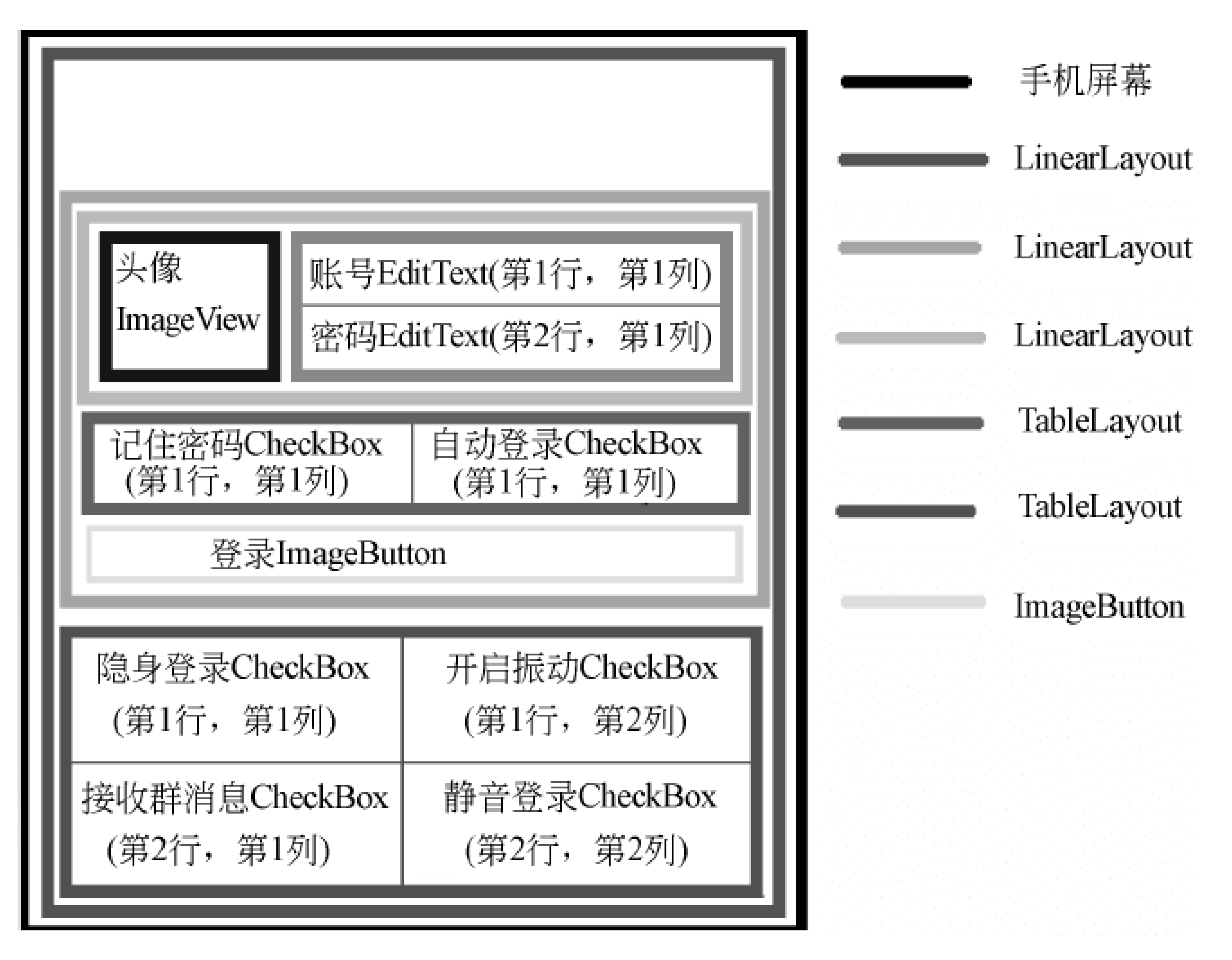


图 3.16　多布局的嵌套使用

实现上面效果的 XML 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout height="fill parent" android:background="@drawable/
    loginbackground">
    <!-- 登录上半部分界面布局-->
    <LinearLayout android:orientation="vertical"
        android:layout width="300px" android:layout height="200px"
        android:background="@drawable/loginbg" android:layout marginTop=
        "51px"
        android:layout_marginLeft="10px" android:layout_marginRight="10px">
        <LinearLayout android:orientation="horizontal"
            android:layout width="match parent" android:layout height=
            "wrap content"
            android:layout_marginLeft="10px" android:layout_marginRight=
            "10px"
            android:layout marginTop="10px">
            <!-- QQ 头像图片 -->
            <!-- ImageView:图片组件，用来显示图片的。该组件的详细属性使用会在第 4
            章中详细讲解 -->
            <ImageView android:layout_width="wrap_content"
                android:layout height="wrap content" android:background=
                "@drawable/qqpic" />
            <!-- 账号密码文本框  -->
            <TableLayout android:layout_width="wrap_content"
                android:layout height="wrap content">
                <!-- 第一行 -->
                <TableRow>
                    <!--EditText:文本框组件，android:singleLine="true": 表示
                    该文本框为单行
                        该组件的详细属性使用会在第 4 章中详细讲解-->
                    <EditText android:hint="账号" android:layout_width=
                    "210px"
                        android:layout_height="wrap_content" android:
```

```
                        singleLine="true" />
                </TableRow>
                <!-- 第二行 -->
                <TableRow>
                    <!-- android:password="true":表示该文本框是用来输入密码的，
                    输入的字符会以某种回显字符方式显示 -->
                    <EditText android:hint="密码" android:layout width=
                    "210px"
                        android:layout_height="wrap_content" android:
                        singleLine="true"
                        android:password="true" />
                </TableRow>
            </TableLayout>
        </LinearLayout>
        <!-- 记住密码和自动登录复选框的 TableLayout 布局 -->
        <TableLayout android:layout width="wrap content"
            android:layout height="wrap content" android:layout marginLeft=
            "10px">
            <!-- 第一行 -->
            <TableRow>
                <!--CheckBox:复选框组件，android:checked="true": 默认选中，该
                组件的详细属性使用会在第 4 章中详细讲解   -->
                <CheckBox android:checked="true" android:text="记住密码"
                    android:textColor="#FF000000" />
                <CheckBox android:checked="true" android:layout_marginLeft=
                "68px"
                    android:textColor="#FF000000" android:text="自动登录" />
            </TableRow>
        </TableLayout>
        <!-- 登录按钮 -->
        <!-- ImageButton:图片按钮，该按钮上面可以显示图片，该组件的详细属性使用会
        在第 4 章中详细讲解 -->
        <ImageButton android:layout width="164px"
            android:layout height="35px" android:layout gravity="center"
            android:background="@drawable/loginbutt" />
    </LinearLayout>
    <!-- 界面底部的四个复选框的 TableLayout 布局-->
    <TableLayout android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:layout_marginLeft=
        "10px"
        android:layout marginTop="40px">
        <TableRow>
            <CheckBox android:text="隐身登录" android:textColor="#FF000000" />
            <CheckBox android:layout marginLeft="68px"
                android:textColor="#FF000000" android:text="开启振动" />
        </TableRow>
        <TableRow>
            <CheckBox android:text="接收群消息" android:textColor="#FF000000" />
            <CheckBox android:layout_marginLeft="68px"
                android:textColor="#FF000000" android:text="静音登录" />
        </TableRow>
    </TableLayout>
</LinearLayout>
```

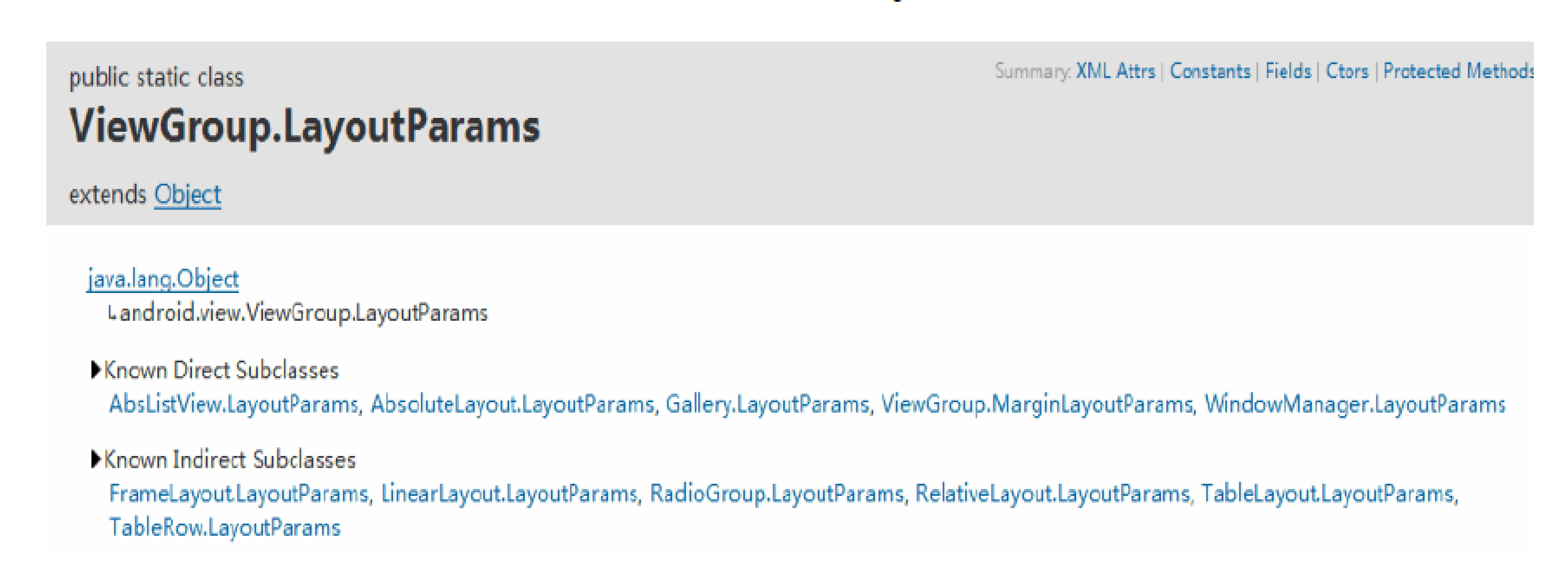
运行效果如图 3.17 所示。



图 3.17　QQ 登录布局

3.6　使用代码完成自定义布局

通过前面几节的学习，我们知道了如何通过 XML 实现界面的布局，但有时候我们往往需要动态地去生成一个布局，这时候就需要在代码中实现布局。LayoutParams 类是用于子视图告诉父视图它的摆放位置，如图 3.18 为 LayoutParams 类的层级图。


Summary: XML Attrs | Constants | Fields | Ctors | Protected Methods

public static class
ViewGroup.LayoutParams
extends Object

java.lang.Object
  ↳android.view.ViewGroup.LayoutParams

▸Known Direct Subclasses
AbsListView.LayoutParams, AbsoluteLayout.LayoutParams, Gallery.LayoutParams, ViewGroup.MarginLayoutParams, WindowManager.LayoutParams

▸Known Indirect Subclasses
FrameLayout.LayoutParams, LinearLayout.LayoutParams, RadioGroup.LayoutParams, RelativeLayout.LayoutParams, TableLayout.LayoutParams, TableRow.LayoutParams

图 3.18　LayoutParams 类层级图

LayoutParams 的基类中提供了常量 FILL_PARENT，MATCH_PARENT，WRAP_CONTENT 用于为不同尺寸的视图指定宽度和高度，对于不同的 ViewGroup 提供了不同的 LayoutParams 子类，例如，AbsoluteLayout 布局有自己的 LayoutParams 子类，可以指定宽度、高度和具体的 x，y 坐标值。下面我们实现在一个流式布局中，通过代码在界面中添加一个 TextView，并通过 LayoutParams 设置 TextView 的布局参数。

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 流式布局，id 为 layout-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:id="@+id/layout"
    android:background="#FFF0F0F0">
</LinearLayout>
```

LayoutParams1 .java 代码如下：

```
package com.LayoutParams1;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class LayoutParams1 extends Activity {
    /** Called when the activity is first created. */
    LinearLayout  liearLayout;                          //声明 LinearLayout 变量
    @Override
    public void onCreate(Bundle savedInstanceState)  {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);                  //加载 main.xml 布局文件
        //获取 main.xml 中 id 为 layout 的 liearLayout 对象
        liearLayout = (LinearLayout) this.findViewById(R.id.layout);
        TextView txtFont = new TextView(this);          //声明一个 TextView 对象
        //声明 ViewGroup.LayoutParams 对象
        ViewGroup.LayoutParams layoutParams = new ViewGroup.LayoutParams(
        ViewGroup.LayoutParams.FILL_PARENT,             //设置布局参数对象的宽度
                ViewGroup.LayoutParams.WRAP_CONTENT);
                                                        //设置布局参数对象的高度
        txtFont.setLayoutParams(layoutParams);
                                    //将 txtFont 以 layoutParams 对象指定宽度和高度布局
        txtFont.setText("通过代码实现布局示例 1"); //设置 txtFont 上面显示的文字
        txtFont.setTextColor(Color.BLACK);          //设置字体颜色为黑色
        liearLayout.addView(txtFont);   //将 txtFont 添加到 LiearLayout 布局上
    }
}
```

运行效果如图 3.19 所示。

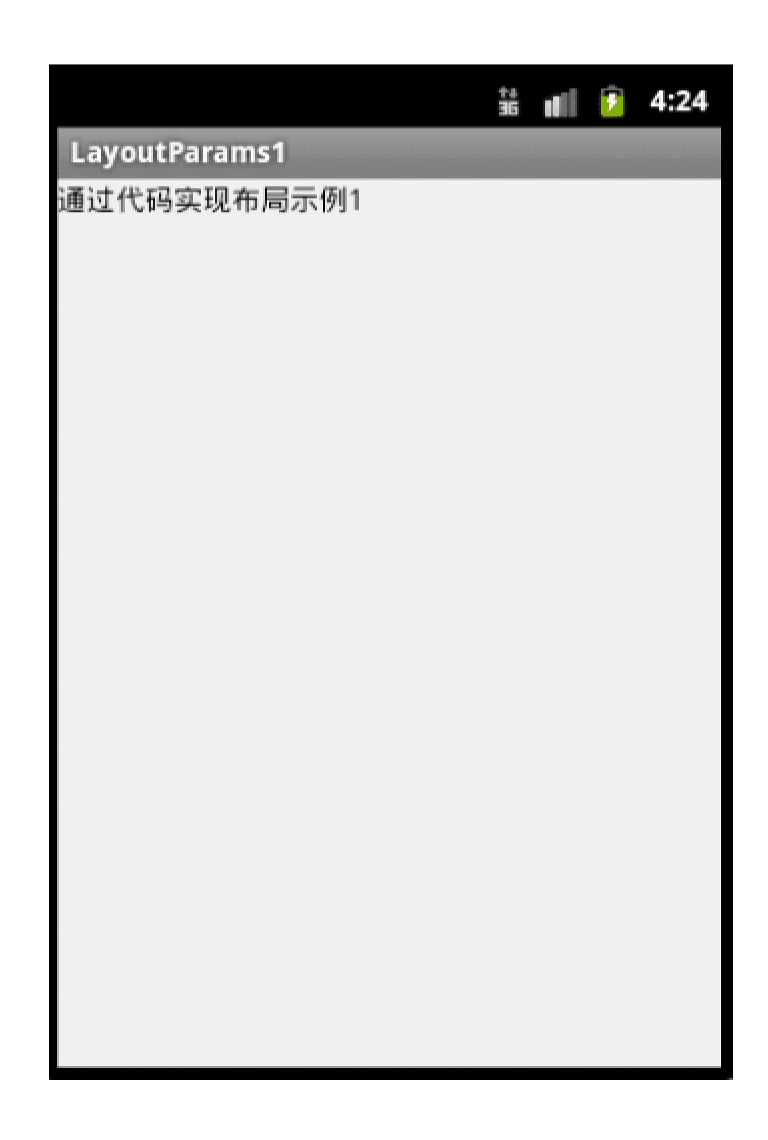


图 3.19　LayoutParams1 示例

在图 3.19 中，TextView 组件及布局是通过代码生成的，代码如下：

```
TextView txtFont = new TextView(this);          //声明一个 TextView 对象
//声明 ViewGroup.LayoutParams 对象
ViewGroup.LayoutParams layoutParams = new ViewGroup.LayoutParams(
        ViewGroup.LayoutParams.FILL_PARENT,     //设置布局参数对象的宽度
        ViewGroup.LayoutParams.WRAP_CONTENT);   //设置布局参数对象的高度
        txtFont.setLayoutParams(layoutParams);
```

```
                              //将 txtFont 以 layoutParams 对象指定宽度和高度布局
        txtFont.setText("通过代码实现布局示例 1"); //设置 txtFont 上面显示的文字
        txtFont.setTextColor(Color.BLACK);     //设置字体颜色为黑色
```

这段代码等价于下面的 XML 代码：

```
<TextView android:layout width="fill parent"
        android:layout height="wrap content" android:text="通过代码实现布局
        示例 1"
        android:textColor="#FF000000" />
```

下面的例子实现较为复杂，LinearLayout.LayoutParams 类中提供了 void setMargins(int left, int top, int right, int bottom)方法可以设置组件左、上、右、下的边距。

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
    <!-- 流式布局，id 为 layout-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout height="fill parent" android:id="@+id/layout"
    android:background="#FFF0F0F0" android:padding="8px">
</LinearLayout>
```

LayoutParams2 .java 代码如下：

```
package com.LayoutParams2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class LayoutParams2 extends Activity {
    /** Called when the activity is first created. */
    LinearLayout liearLayoutMain;              //声明 LinearLayout 变量
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);          //加载 main.xml 布局文件
        //获取 main.xml 中 id 为 layout 的 liearLayout 对象
        liearLayoutMain = (LinearLayout) this.findViewById(R.id.layout);
        TextView bookName = new TextView(this);//声明一个 TextView 对象
        //声明 ViewGroup.LayoutParams 对象
        ViewGroup.LayoutParams bookNameParams = new ViewGroup.LayoutParams(
        ViewGroup.LayoutParams. FILL_PARENT,    //设置布局参数对象的宽度
                ViewGroup.LayoutParams.WRAP CONTENT);
                                                //设置布局参数对象的高
        bookName.setLayoutParams(bookNameParams);//设置 bookName 的布局方式
        bookName.setText("桂林上水甲天下");      //设置 bookName 上面显示的文字
        bookName.setTextColor(Color.BLACK);     //设置 bookName 的文字颜色
        bookName.setGravity(Gravity.CENTER);    //设置 bookName 文字居中显示
        TextPaint bttp = bookName.getPaint();   //获取 TextPaint 对象
        bttp.setFakeBoldText(true);             //设置 bookName 加粗显示

        TextView bookDesc = new TextView(this);//声明一个 TextView 对象
        //声明 ViewGroup.LayoutParams 对象
        LinearLayout.LayoutParams bookDescParams = new LinearLayout.
        LayoutParams(
                ViewGroup.LayoutParams.WRAP_CONTENT,//设置布局参数对象的宽度
                ViewGroup.LayoutParams.WRAP_CONTENT);//设置布局参数对象的高度
        bookDesc.setLayoutParams(bookDescParams);//设置 bookDesc 的布局方式
```

```
        bookDesc.setText(R.string.desc);        //设置 txtFont 上面显示的文字
        bookDesc.setTextColor(Color.BLACK);

        ImageView bookPic = new ImageView(this);//声明一个 ImageView 对象
        LinearLayout.LayoutParams bookPicParams = new LinearLayout.
        LayoutParams(
                251, 183);
        //设置边距，方法原型 void setMargins (int left, int top, int right,
        int bottom)
        bookPicParams.setMargins(25, 0, 0, 5);
        bookPic.setLayoutParams(bookPicParams);
        bookPic.setBackgroundResource(R.drawable.guilin);
                                    //设置 ImageView 的背景图

        liearLayoutMain.addView(bookName);
                                    //将 bookName 添加到 liearLayoutMain 布局上
        liearLayoutMain.addView(bookPic);
                                    //将 bookPic 添加到 liearLayoutMain 布局上
        liearLayoutMain.addView(bookDesc);
                                    //将 bookDesc 添加到 liearLayoutMain 布局上
    }
}
```

res/values/strings.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="desc">简介：桂林旅游资源丰富，并具有以下五个方面特点和优势：一是山川秀丽独特。典型的喀斯特地貌和丹霞地貌赋予桂林“山青、水秀、洞奇、石美”的山水风光，素有“甲天下”的盛誉；二是自然资源和人文资源浑然一体。漓江、芦笛岩、象鼻山、龙脊、八角寨、灵渠、桂海碑林、王城等景区（点）各具特色；三是城市与景区交融，推窗、出门就能见景，且景观分布适当，旅游可达性好；四是景观分布的空间层次多，且各具特色，便于多种旅游线路的组合与分期分区开发；五是多数景区有城镇做依托。</string>
    <string name="app_name">LayoutParams1</string>
</resources>
```

运行效果如图 3.20 所示。



图 3.20　LayoutParams2 示例

第 4 章　Android 人机界面

通过前一章的学习，我们对 Android 的布局管理器已有所了解。Android 中还提供了丰富多彩的界面组件，这些组件让界面变得华丽起来。对于组件的生成可以通过 XML 实现，可以通过 Java 代码来实现，也可以通过 DroidDraw 在图形化界面上创建。现在我们就开始走入 Android 丰富多彩的 UI 世界吧。

4.1　全屏显示——标题、状态栏的隐藏

Android 中提供了 Window 类，用于设置窗口的属性和基本功能；Activity 类中提供了一个方法 public final boolean requestWindowFeature (int featureId)，用于设置 Window 的属性，参数 featureId 取值由 Window 类定义。Window 常用属性如下表 4.1 所示。

表 4.1　Window常量值列表

常　量　名	常量值	功　　能
FEATURE_CONTEXT_MENU	6	上下文菜单，默认值
FEATURE_CUSTOM_TITLE	7	自定义标题栏，该属性不能和其他标题栏属性同时使用
FEATURE_LEFT_ICON	3	在标题栏左侧显示图标
FEATURE_RIGHT_ICON	4	在标题栏右侧显示图标
FEATURE_PROGRESS	2	在标题栏上显示进度条
PROGRESS_VISIBILITY_ON	-1	进度条可见
PROGRESS_VISIBILITY_OFF	-2	进度条不可见
PROGRESS_START	0	第一进度条的最小值
PROGRESS_END	10000	第一进度条的最大值
PROGRESS_SECONDARY_START	20000	第二进度条的最小值
PROGRESS_SECONDARY_END	30000	第二进度条的最大值
FEATURE_NO_TITLE	1	无标题栏

下面的例子实现了窗口标题栏和状态栏的隐藏。WindowExample .java 代码如下：

```
package com.WindowExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class WindowExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Window window=this.getWindow();          //获取当前 Activity 的 Window
```

```
        //隐藏窗体的状态栏
        window.setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN);
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);
                                              //隐藏窗体上方的标题栏
        setContentView(R.layout.main);

    }
}
```

注意：requestWindowFeature()方法要在 setContentView()方法之前使用。

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFF0F0F0">
    <!-- 标签组件，文字颜色为黑色，文字内容为 res/values/strings.xml 中 hello 变量
    中的内容-->
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="@string/hello"
        android:textColor="#FF000000" />
</LinearLayout>
```

运行效果如图 4.1 所示。

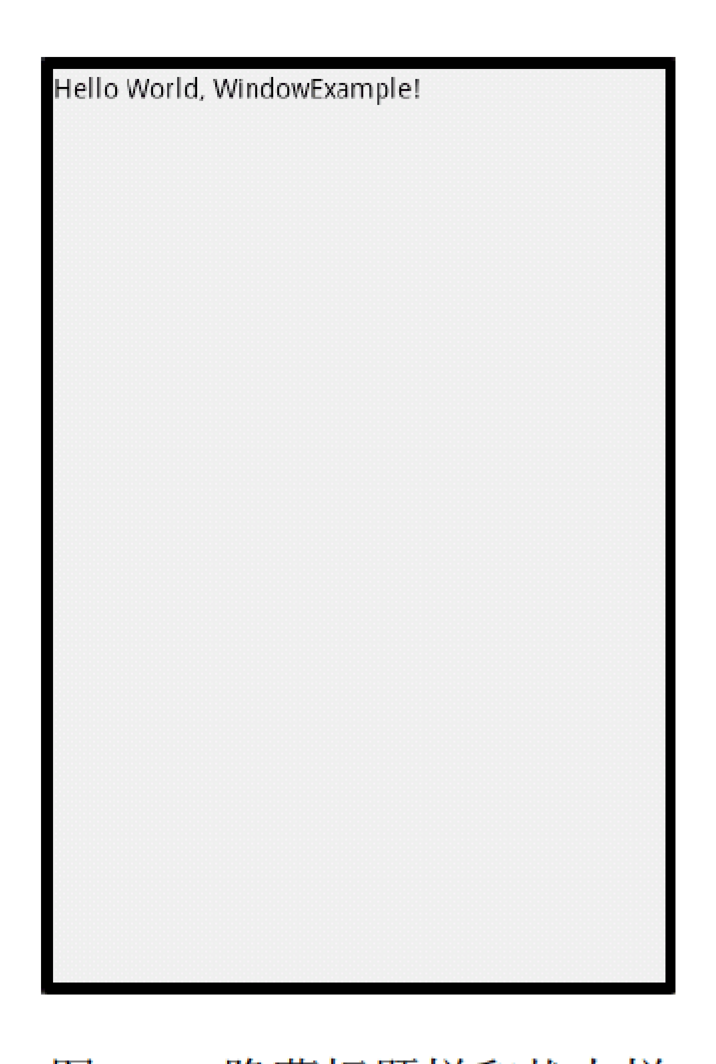


图 4.1　隐藏标题栏和状态栏

再看下面的这个例子，在标题栏上显示进度条。main.xml 内容不变，修改 WindowExample.java 代码如下：

```
package com.WindowExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class WindowExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setTitle("progressing....");        //设置标题栏上的文字
```

```
        Window window=this.getWindow();             //获取当前 Activity 的 Window
        this.requestWindowFeature(Window.FEATURE_PROGRESS);
                                                    //标题栏显示滚动条
        setContentView(R.layout.main);              //设置当前窗体的布局管理文件
        this.setProgressBarVisibility(true);        //设置进度条可见
        this.setProgress(1800);                     //设置第一进度条的长度
        this.setSecondaryProgress(8000);            //设置第二进度条的长度

    }
}
```

运行效果如图 4.2 所示。



图 4.2　标题栏显示进度条

同样可以设置标题栏上显示图标，图标文件为 res/drawable/webicon.png，上面的 res/layout/main.xml 文件内容不变，修改源文件如下：

```
package com.WindowExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class WindowExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setTitle("Window Icon");               //设置标题栏上的文字
        Window window=this.getWindow();             //获取当前 Activity 的 Window
        this.requestWindowFeature(Window.FEATURE_LEFT_ICON);
                                                    //标题栏左侧显示图标
        setContentView(R.layout.main);              //设置当前窗体的布局管理文件
        //设置左侧的图标
        this.setFeatureDrawableResource(Window.FEATURE_LEFT_ICON, R.drawable.
        webicon);
    }
}
```

运行效果如图 4.3 所示。



图 4.3　标题栏显示图标

4.2　样式化的定型对象——style 的使用

style 类似于 CSS，CSS 是将 HTML 的标签属性剥离出来，统一放到 CSS 样式文件中，这样的优点是方便修改和管理代码。style 是将 TextView、Button 等 Android 组件的 XML 属性剥离出来，统一放到 XML 文件中，这些剥离出来的属性放到<style>标签中，每一个属性是一个单独的<item>，一个<style>标签中可以有一个或者多个<item>。当需要加载某一个 style 时，只需要通过<style>的 name 值去调用该 style。下面我们看看如何通过 style 去定义 TextView 的字体、颜色等属性值。

StyleExample1 .java 代码如下：

```
package com.StyleExample1;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class StyleExample1 extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);            //加载布局
    }
}
```

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#ffF4F4F4">
    <TextView style="@style/DefineTextStyle" android:layout_width="fill_
    parent"
        android:layout_height="wrap_content" android:text="@string/hello"
```

```
        />
</LinearLayout>
```

代码说明：

- TextView 标签的 style 属性用于加载自定义的 style。这里加载名字为 DefineTextStyle 的 style，该 style 定义在 style.xml 文件中。
- TextView 标签的 text 属性指定标签上显示的文字为字符串 hello 内存储的值，字符串 hello 的值可以在 strings.xml 中看到。

res/values/ strings.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Android World!</string>
    <string name="app_name">style 的使用</string>
    <color name="pink">#FFC8FF</color>
    <color name="darkgreen">#008800</color>
</resources>
```

代码说明：

- 一般在 strings.xml 中定义项目中用到的常量。
- string 标签用来定义字符串常量，项目中用到 string 标签中的字符串，可以通过名字引用的方式使用该字符串。
- Color 标签用来定义颜色常量，和 string 同理。

res/values/ style.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="DefineTextStyle">
        <item name="android:textSize">28px</item>
        <item name="android:textColor">@color/darkgreen</item>
        <item name="android:gravity">center</item>
        <item name="android:textStyle">bold|italic
        </item>
        <item name="android:background">@color/pink
        </item>
    </style>
</resources>
```

代码说明：

- style 标签用来定义自定义的 XML 属性。通过 style 标签的名字来使用该 style。可以定义多个 style。
- item 标签定义每一个 XML 属性值。在 style 标签中可以有 0 个或多个 item 标签。

运行效果如图 4.4 所示。



图 4.4　style 的使用

4.3　玩转 TextView——标签特效

TextView 是最常用的组件之一，用来显示窗口上的文字信息。如图 4.5 所示，TextView 是 View 的子类，TextView 继承了 View 的属性和方法。

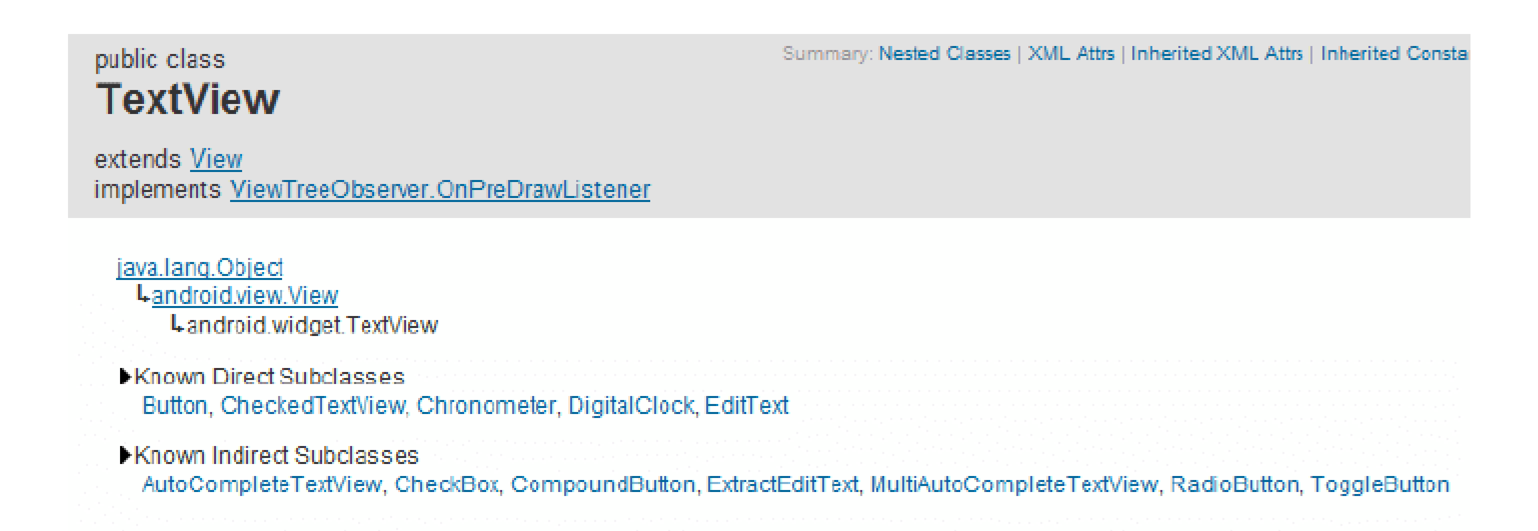


图 4.5　EditText 与 TextView 的类层级图

View 的常用属性及其含义见表 4.2。

表 4.2　View的XML属性

属 性 名 称	相 关 方 法	属 性 描 述
android:background	setBackgroundResource(int)	设置组件的背景颜色或者背景图片
android:fadingEdge	setVerticalFadingEdgeEnabled (boolean)	设置拉动滚动条时，边框渐变的方向，有三种取值。none：边框颜色不变；vertical：垂直方向颜色变淡；horizontal：水平方向颜色变淡
android:fadingEdgeLength	getVerticalFadingEdgeLength()	设置边框渐变的长度
android:id	setId(int)	设置组件的 id 值
android:focusable	setFocusable(boolean)	设置是否获取焦点，取值是 true 或者 false，默认是 false
android:focusableIn TouchMode	setFocusableInTouchMode (boolean)	在 touch 模式下是否获取焦点
android:padding	setPadding(int,int,int,int)	设置组件的上下左右边距，以像素为单位
android:paddingBottom	setPadding(int,int,int,int)	设置组件的底部边距，以像素为单位
android:paddingLeft	setPadding(int,int,int,int)	设置组件的左侧边距，以像素为单位
android:paddingRight	setPadding(int,int,int,int)	设置组件的右侧边距，以像素为单位
android:paddingTop	setPadding(int,int,int,int)	设置组件的顶部边距，以像素为单位
android:scrollbarThumb Horizontal		设置水平滚动条的 drawable
android:scrollbarThumb Vertical		设置垂直滚动条的 drawable

下面我们应用一下 View 的属性，实现窗口滚动效果，并设置滚动条的背景图片。

ViewExample .ava 代码如下：

```
package com.ViewExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ViewExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);            //加载布局文件
    }
}
```

res/values/strings.xml 内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
```

```
    <string name="hello">Hello World, TextViewExample1!</string>
    <string name="lineheight1">设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，
    设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条，设置滚动条
    </string>
    <string name="app name">ScrollExample</string>
</resources>
```

滚动条的背景图片存储位置为 res/drawable/scrollbackground.png

res/layout/main.xml 内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
    <!--
        ScrollView 标签为滚动 view 标签，当内容超出窗口可视范围会自动出现滚动条
        android:scrollbarThumbVertical:  指定滚动条的背景图片
    -->
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout height="fill parent" android:background="#ffF4F4F4"
    android:fadingEdge="vertical" android:fadingEdgeLength="100px"
    android:scrollbarThumbVertical="@drawable/scrollbackground">
    <!-- android:text:指定 TextView 上显示的文字为 strings.xml 中变量 lineheight1
    内容  -->
    <TextView android:layout width="fill parent"
        android:layout height="wrap content" android:text="@string/lineheight1"
        android:textColor="#ff000000" />
</ScrollView>
```

运行效果如图 4.6 所示。



图 4.6　窗口的滚动效果

TextView 的常用属性及其含义，见表 4.3。

表 4.3　TextView的XML属性

属 性 名 称	相 关 方 法	属 性 描 述
android:autoLink	setAutoLinkMask(int)	设置是否当文本为 URL 连接/电话号码/email/map 时，文本显示为可单击的链接。可选值为 none/web/email/phone/map/all
android:ellipsize	setEllipsize (TextUtils.TruncateAt)	设置文字过长时，组件如何显示文字，取值如下。none：文字过长时，正常显示，即不处理，也是默认值。start：省略号显示在开头。 middle：省略号显示在中间。end：省略号显示在结尾。marquee：以跑马灯方式显示，注意如为跑马灯方式，需要设置 android:focusable="true" android:focusableInTouchMode="true"才有效果
android:gravity	setGravity(int)	设置文本在组件中显示的位置，设置为 left 为文本居左显示，设置为 right 为文本居右显示，设置为 center 为文本居中显示
android:hint	setHint(int)	Text 为空时的文字提示信息，此属性一般在 EditView 中使用，这里也可以使用
android:lineSpacingExtra	setLineSpacing(float,float)	设置行间距
android:lineSpacing Multiplier	setLineSpacing(float,float)	设置行间距的倍距，如 1.5
android:marquee RepeatLimit	setMarqueeRepeatLimit(int)	当 ellipsize 值为 marquee 时，设置重复滚动的次数，当设置为 marquee_forever 时表示无限次
android:shadowColor	setShadowLayer (float,float,float,int)	设置文本阴影的颜色，一般和 shadowRadius 一起使用
android:shadowRadius	setShadowLayer (float,float,float,int)	设置阴影半径，如设置为 0.1 则为字体的颜色，一般设置为 3.0 效果较好
android:singleLine	setTransformationMethod (TransformationMethod)	设置文本是否单行显示，如果和 layout_width 一起使用，当文本不能全部显示时，后面用“…”表示
android:text	setText(CharSequence, TextView.BufferType)	设置组件上显示的文本
android:textColor	setTextColor(ColorStateList)	设置文本颜色
android:textColor Highlight	setHighlightColor(int)	设置被选中文字的底色，默认为蓝色
android:textColorHint	setHintTextColor (ColorStateList)	设置 hint 提示文字的颜色
android:textColorLink	setLinkTextColor(int)	设置超链接文字的颜色
android:textScaleX	setTextScaleX(float)	设置文字缩放，默认为 1.0
android:textSize	setTextSize(float)	设置文字的大小。推荐使用单位“sp”
android:textStyle	setTypeface(Typeface)	设置字形，可选值为 bold：粗体，italic：斜体，normal：正常。可以同时设置多个值用“|”隔开
android:password	setTransformationMethod (TransformationMethod)	设置回显字符，以点“.”的方式显示

通过下面的例子，我们看看 TextView 都可以实现哪些效果。

TextViewExample .java 代码如下：

```
package com.TextViewExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class TextViewExample extends Activity {
    /** Called when the activity is first created. */
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);          //加载布局文件
    }
}
```

res/values/strings.xml 内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, TextViewExample1!</string>
    <string name="lineheight1">设置固定行间距，设置固定行间距，设置固定行间距，设
    置固定行间距，设置固定行间距，设置固定行间距，设置固定行间距，设置固定行间距</string>
     <string name="lineheight2">1.5 倍行间距,1.5 倍行间距,1.5 倍行间距,1.5 倍行
     间距,1.5 倍行间距,1.5 倍行间距,1.5 倍行间距,1.5 倍行间距,1.5 倍行间距,1.5 倍行间
     距</string>
    <string name="txtlink">文字超链接：<a href='http://www.baidu.com'>首页
     </a></string>
    <string name="tellink">电话超链接：<a href='http://www.baidu.com'>110
     </a></string>
    <string name="app name">TextViewExample</string>
</resources>
```

res/layout/main.xml 内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout height="fill parent" android:background="#ffF4F4F4"
    android:padding="8px">
    <!-- 文字居中 -->
    <TextView android:id="@+id/testGravity" android:layout_width="fill_
    parent"
        android:layout_height="wrap_content" android:text="居中文字"
        android:gravity="center" android:background="#ff00ff00"
        android:textColor="#ff000000" android:textSize="18sp" />
    <!-- 文字跑马灯效果 -->
    <TextView android:id="@+id/testEllipsize"
        android:layout_width="100px" android:layout_height="wrap_content"
        android:text="跑马灯文字效果" android:ellipsize="marquee"
        android:marqueeRepeatLimit="marquee forever" android:singleLine=
        "true"
        android:focusable="true" android:focusableInTouchMode="true"
        android:textColor="#ff000000" android:textSize="20sp" />
    <!-- 设置文字阴影效果 -->
    <TextView android:id="@+id/testShadow" android:layout_width="fill_
    parent"
        android:layout_height="wrap_content" android:text="文字阴影效果"
        android:textColor="#ff000000" android:shadowColor="#ffff0000"
        android:shadowRadius="3.0" />
    <!-- 设置网址超链接效果 -->
    <TextView android:id="@+id/testAutoLink1"
        android:layout_width="fill_parent" android:layout_height="wrap_
        content"
        android:text="网址超链接：http://www.baidu.com" android:textColor=
        "#ff000000"
        android:autoLink="all" android:textColorLink="#ff0000ff" />
    <!-- 设置文字超链接效果 -->
    <TextView android:id="@+id/testAutoLink2"
```

```
        android:layout width="fill parent" android:layout height="wrap
        content"
        android:text="@string/txtlink" android:textColor="#ff000000" />
    <!-- 设置电话超链接效果 -->
    <TextView android:id="@+id/testAutoLink2"
        android:layout_width="fill_parent" android:layout_height="wrap_
        content"
        android:text="@string/tellink" android:textColor="#ff000000" />
    <!-- 设置字形 -->
    <TextView android:id="@+id/testTextStyle"
        android:layout width="fill parent" android:layout height="wrap
        content"
        android:text="斜体" android:textColor="#ff000000" android:textStyle=
        "italic" />
    <!-- 设置文字缩放 -->
    <TextView android:id="@+id/testTextScaleX"
        android:layout width="fill parent" android:layout height="wrap
        content"
        android:text="hello 0.5f" android:textColor="#ff000000"
        android:textScaleX="0.5" />
    <TextView android:layout width="fill parent"
        android:layout height="wrap content" android:text="hello 1.0f"
        android:textColor="#ff000000" android:textScaleX="1.0" />
    <TextView android:layout width="fill parent"
        android:layout height="wrap content" android:text="hello 1.5f"
        android:textColor="#ff000000" android:textScaleX="1.5" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="hello 2.0f"
        android:textColor="#ff000000" android:textScaleX="2.0" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="hello 2.5f"
        android:textColor="#ff000000" android:textScaleX="2.5" />
    <!-- 设置行间距 -->
    <TextView android:id="@+id/testLineSpacingExtra"
        android:layout width="fill parent" android:layout height="wrap
        content"
        android:text="@string/lineheight1" android:textColor="#ff000000"
        android:lineSpacingExtra="4px" />
    <!-- 设置行间距的倍数 -->
    <TextView android:id="@+id/testLineSpacingMultiplier"
        android:layout_width="fill_parent" android:layout_height="wrap_
        content"
        android:text="@string/lineheight2" android:textColor="#ff000000"
        android:lineSpacingMultiplier="1.5" />
</LinearLayout>
```

运行效果如图 4.7 所示。

上面 TextView 的属性设置可以在 XML 文件中设置，也可以通过代码来实现，如用文字居中效果，TextView 中提供了方法 public void setGravity(int gravity)，参数为 int 类型，为组件内文字的对齐方式，Gravity 类中提供了对齐方式的常量值。

图 4.7 示例中的文字居中效果也可以通过如下代码实现：

```
TextView testGravity=(TextView) this.findViewById(R.id.
testGravity);                //获取 TextView 组件
testGravity.setGravity(Gravity.CENTER);
                    //设置 TextView 文字对齐方式为居中对齐
```

图 4.7　TextView 示例

图 4.7 示例中跑马灯效果可以通过下面代码实现：

```
TextView testEllipsize=(TextView) this.findViewById(R.id.testEllipsize);
//获取 TextView 对象
//设置文字过长时为跑马灯效果，在 TruncateAt 类中定义了常量值
testEllipsize.setEllipsize(TruncateAt.MARQUEE);
testEllipsize.setMarqueeRepeatLimit(-1);        //-1 表示设置跑马灯为无限次
testEllipsize.setFocusable(true);               //设置组件获取焦点
testEllipsize.setFocusableInTouchMode(true);    //在 touch 模式下获取焦点
//设置文字单行显示，如不设置文字会被挤到下一行，看不到跑马灯效果
testEllipsize.setSingleLine(true);
```

TextView 的超链接效果可以通过以下 4 种方式来实现。

方式一：在 XML 中使用属性 android:autoLink 属性，代码如下。

```
<!-- 设置网址超链接效果 -->
<TextView android:id="@+id/testAutoLink1"
android:layout width="fill parent" android:layout height="wrap content"
    android:text="网址超链接：http://www.baidu.com" android:textColor=
    "#ff000000"
    android:autoLink="all" android:textColorLink="#ff0000ff" />
```

方式二：通过在文本中使用<a>标签，代码如下。

```
<string name="tellink">电话超链接：<a href='http://www.baidu.com'>110</a>
</string>
```

方式三：通过 fromHtml 方法，和第二种方法类似，只是这种方法将文本写到了 Java 代码中，代码如下。

```
TextView testAutoLink1=(TextView) this.findViewById(R.id.testAutoLink1);
//获取 TextView 组件
testAutoLink1.setText(Html.fromHtml("网址超链接：<a href='http://www.baidu.
com'>http://www.baidu.com</a>"));
testAutoLink1.setLinkTextColor(Color.BLUE); //设置超链接的颜色为蓝色，Color
类中定义了颜色常量值
//setMovementMethod 方法用于响应单击超链接时响应的用户事件，如不执行该方法，则即使
看到超链接效果也不响应事件，如单击电话号码，跳到拨号的页面
testAutoLink1.setMovementMethod(LinkMovementMethod.getInstance());
```

方式四：通过 SpannableString 创建字符串，通过 setSpan 方法设置一个或者多个超链接，代码如下。

```
TextView testAutoLink1=(TextView)findViewById(R.id.testAutoLink1);
SpannableString ss= new SpannableString("Click here to dail the phone");
//setSpan 设置 SpannableString 字符串中的那部分字符为超链接效果，需要四个参数，含义
分别是
//参数 1：设置 Span 字符特效,这里可以设置超链接，粗体，斜体，以及文字背景前景等特效
//参数 2：设置 SpannableString 字符串中字符特效文字的起始位置，第一个字符位置为 0
//参数 3：设置 SpannableString 字符串中字符特效文字的终止位置
//参数 4：设置 Span 范围内的文本前后输入新的字符时是否把它们也应用这个效果，可选值为
//Spanned.EXCLUSIVE_EXCLUSIVE：前后都不包括
//Spanned.SPAN_INCLUSIVE_INCLUSIVE:前后都包括
//Spanned.SPAN_EXCLUSIVE_INCLUSIVE:前面不包括，后面包括
//Spanned.SPAN_INCLUSIVE_EXCLUSIVE:前面包括，后面不包括
```

```
//URLSpan 为设置为超链接特效
ss.setSpan(new URLSpan("tel:13143234567"),6, 10, Spanned.SPAN_EXCLUSIVE_
EXCLUSIVE);
//StyleSpan 为设置字体特效，Typeface.ITALIC 为斜体
ss.setSpan(new StyleSpan(Typeface.ITALIC), 0, 5, Spanned.SPAN_EXCLUSIVE_
EXCLUSIVE);
//Typeface.BOLD 为粗体
ss.setSpan(new StyleSpan(Typeface.BOLD), 23, ss.length(), Spanned.SPAN_
EXCLUSIVE_EXCLUSIVE);
//BackgroundColorSpan 设置背景颜色
ss.setSpan(new BackgroundColorSpan(0xFFFFFF00), 14, 18, Spanned.SPAN_
EXCLUSIVE_EXCLUSIVE);
//ForegroundColorSpan 设置前景颜色
ss.setSpan(new ForegroundColorSpan(Color.RED),  0, 5, Spanned.SPAN_
EXCLUSIVE_EXCLUSIVE);
//StrikethroughSpan 设置为删除线
ss.setSpan(new StrikethroughSpan(), 11, 13,Spanned.SPAN_EXCLUSIVE_
EXCLUSIVE);
//AbsoluteSizeSpan 设置字体大小
ss.setSpan(new AbsoluteSizeSpan(25), 6, 10, Spanned.SPAN_EXCLUSIVE_
EXCLUSIVE);testAutoLink1.setText(ss);
testAutoLink1.setMovementMethod(LinkMovementMethod.getInstance());
```

运行效果如图 4.8 所示。



图 4.8　SpannableString 示例

图 4.7 中行间距及行间距的倍距也可以通过下面代码实现：

```
TextView testLineSpacingExtra=(TextView)findViewById(R.id.testLine-
SpacingExtra);  //获取 TextView 组件
testLineSpacingExtra.setLineSpacing(4.0f, 1.0f);
                //设置行间距及倍距，参数 1 是行间距值，参数 2 是行间距倍距
```

4.4　EditText 的使用——文本框

EditText 是文本框组件，用来接收文字的输入。如图 4.9 是 EditText 的类层级图，EditText 继承了 TextView，因此 EditText 可以使用 View 和 TextView 中的属性和方法。



图 4.9　EditText 类层级图

通过下面的示例详细了解 EditText 的使用。EditTextExample .java 代码如下：

```
package com.EditTextExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class EditTextExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);            //加载 main.xml 布局文件
    }
}
```

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout height="fill parent" android:background="#ffF4F4F4"
    android:padding="10px">
    <!--
        EditText 提示信息示例。 android:hint 设置提示信息内容，android:textColor-
        Hint：设置提示信息颜色
    -->
    <EditText android:layout_width="fill_parent"
        android:layout height="wrap content" android:text="" android:
        hint="提示信息"
        android:textColorHint="#FFFF0000" />
    <!--
        EditText 密码及限制输入字符数示例。 android:password：为 true 表示字符以
        点“.”的方式显示
        android:maxLength:设置文本框最多可输入的字符数
    -->
    <EditText android:layout width="fill parent"
        android:layout height="wrap content" android:password="true"
        android:maxLength="6" />
    <!-- EditText 单行显示文本示例。 android:singleLine：无论文本多长都在一行显示-->
    <EditText android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="单行显示单行显示
        单行显示单行显示单行显示单行显示单行显示"
        android:singleLine="true" />
    <!--EditText 多行显示文本示例。  android:lines:设置文本可见行数，类似于 HTML
    的 textarea 文本域效果-->
    <EditText android:layout width="fill parent"
        android:layout height="wrap content"
        android:text="多行显示多行显示多行显示多行显示多行显示多行显示多行显示多行
        显示多行显示多行显示多行显示多行显示多行显示多行显示"
        android:lines="2" />
    <!--
        EditText 文本不可编辑的，相当于 TextView 效果。
        android:enabled 为 false 时为不可编辑，默认值为 true，表示可编辑
    -->
    <EditText android:layout width="fill parent"
        android:layout height="wrap content" android:text="不可编辑的"
        android:enabled="false" />
    <!--EditText 输入电话号码示例，观察软键盘的变化，软键盘变成只能输入电话号码效果，
    这样避免输入一些错误的字符-->
    <EditText android:layout_width="fill_parent"
```

```
    android:layout_height="wrap_content" android:phoneNumber="true"
    android:hint="只接收电话号码" android:textColorHint="#FFFF0000" />
<!--
    EditText 接收数字示例
    。android:numeric 有三种取值：integer:正整数，decimal：浮点数，signed：
    带符号数
-->
<EditText android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:numeric="decimal"
    android:hint="只接收数字" android:textColorHint="#FFFF0000" />
<!--
    为文本指定特定的软键盘。android:inputType 可以指定各种各样的软键盘效果。
    android:imeOptions：设置回车各种各样的显示方式
-->
<EditText android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:inputType="time"
    android:hint="设置软键盘输入方式并设置回车图标效果" android:textColorHint=
    "#FFFF0000"
    android:imeOptions="actionSearch" />
```

</LinearLayout>运行效果如图 4.10 所示。

代码说明如下。

- android:phoneNumber="true"将 EditText 变成只输入电话号码模式，软键盘也变成了拨号专用的软键盘了。避免输入其他错误字符，对校验是否是电话号码就容易多了，效果如图 4.11 所示。



图 4.10　EditText 示例



图 4.11　android:phoneNumber="true"效果图

- android:numeric 用来控制 EditText 只能输入数字。软键盘也变成了数字专用键盘，如图 4.12 所示。
- android:imeOptions 设置回车的图标效果，取值分别如下。
  - normal：默认值，效果为 完成；
  - actionUnspecified：未指定，对应常量为 EditorInfo. IME_NULL，效果为 ↵；
  - actionNone：没有关联动作，对应常量为 EditorInfo. IME_ACTION_NONE，效



图 4.12　android:numeric="decimal"效果图

果为；

- actionGo：去往，对应常量为 EditorInfo.IME_ACTION_GO，效果为；
- actionSearch：搜索，对应常量为 EditorInfo.IME_ACTION_SEARCH，效果为；
- actionSend：发送，对应常量为 EditorInfo.IME_ACTION_SEND，效果为；
- actionNext：下一个，对应常量为 EditorInfo. IME_ACTION_NEXT，效果为；
- actionDone：完成，对应常量为 EditorInfo. IME_ACTION_DONE，效果为。

4.5　简易的按钮事件处理——Button 改变窗体背景及 Drawable 颜色常数介绍

无论是在网站还是在应用程序中，按钮都是很常用的组件之一，是应用程序和用户交互的手段。Android 中的按钮用 Button 类来实现，按钮的事件处理通过 setOnXxxxListener() 方法实现，类似于 Java 中的 Swing 事件处理。Button 类的层次关系如图 4.13 所示。

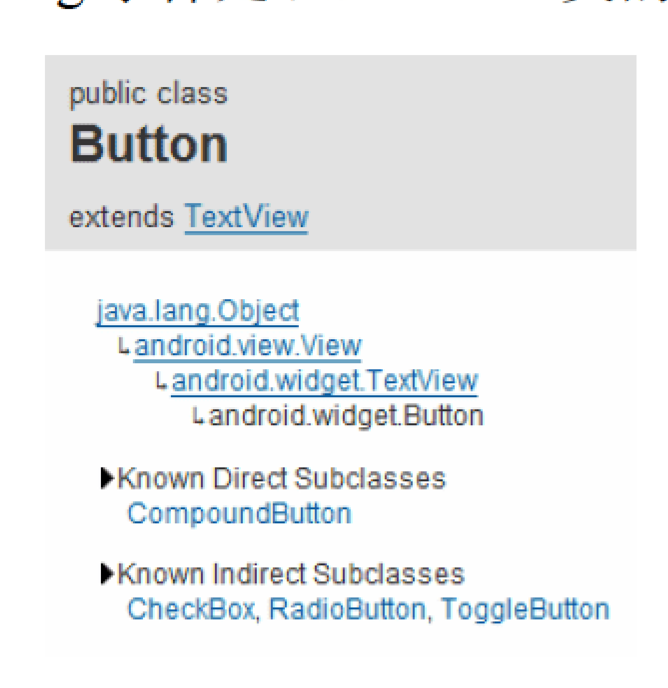


图 4.13　Button 类层级图

下面的例子中，单击不同的按钮，实现改变窗体的背景颜色，现在就了解一下按钮事件触发过程和实现。ButtonEventExample .java 代码如下：

```
package com.ButtonEventExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ButtonEventExample extends Activity {
    private LinearLayout layout;                //声明 LinearLayout 类型变量
    private Button redBut;                      //声明 Button 类型变量
    private Button blueBut;                     //声明 Button 类型变量
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);           //加载 main.xml 布局文件
       layout=(LinearLayout)this.findViewById(R.id.layout);
                                    //通过 id 获取布局文件中的 LinearLayout 对象
       redBut=(Button)this.findViewById(R.id.redBut);
                                    //通过 id 获取布局文件中的 Button 对象
       blueBut=(Button)this.findViewById(R.id.blueBut);
                                    //通过 id 获取布局文件中的 Button 对象

       redBut.setOnClickListener(new OnClickListener(){     //按钮的单击事件
           @Override
```

```
            public void onClick(View v) {       //按钮的单击事件
                //TODO Auto-generated method stub
                layout.setBackgroundColor(Color.RED);
                                                //修改 layout 的背景颜色
                ((Button)v).setText("背景红了"); //修改 Button 按钮上的文字
            }
        });
        blueBut.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                layout.setBackgroundColor(Color.BLUE);
                ((Button)v).setText("背景蓝了"); //修改 Button 按钮上的文字
            }
        });
    }
}
```

代码说明：

Button 类继承了 View 类事件处理的方法，代码中的 setOnClickListener()方法为单击事件，该方法的原型为 public void setOnClickListener (View.OnClickListener l)，参数 View.OnClickListener 是一个接口；该接口中需要实现的抽象方法为 public abstract void onClick (View v)，参数为 View 类型，即当前单击的窗体组件。

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/layout" android:orientation="horizontal"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:background="@drawable/white">
    <Button android:id="@+id/redBut" android:layout width="wrap content"
        android:layout height="wrap content" android:text="红色"
        android:layout gravity="center horizontal" android:layout weight="1"/>
    <Button android:id="@+id/blueBut" android:layout width="wrap content"
        android:layout_height="wrap_content" android:text="蓝色"
        android:layout_gravity="center_horizontal" android:layout_weight="1"/>
</LinearLayout>
```

res/values/strings.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello">Hello World, ButtonEventExample!</string>
   <string name="app_name">ButtonEventExample</string>
   <drawablename="white">#FFFFFFFF</drawable>
</resources>
```

代码说明：

- android:background="@drawable/white"设置窗体的背景颜色为@drawable/white，white 是事先在 strings.xml 定义好的颜色常数，通过这种方式可以修改组件的颜色。修改组件颜色的方式很多，第 2 种方法是通过 color 标签定义颜色，color 格式为 <color name=color_name> color_value</color>。使用该颜色值的时候，通过 Color 标签中指定的 name 名称引用该颜色，和 drawable 用法相同，为 android:background= "@ color / color_name "。

第 3 种方法是直接在属性中赋值，例如 android:background="#FFFFFFFF"。色码值使用十六进制表示，颜色的取值范围为 0～255（00～FF），色码值的顺序为“aarrggbb”，aa：alpha，为透明度，取值为 00～FF；rr：red，取值为 00～FF；gg：green，取值为 00～FF；bb：blue，取值为 00～FF。

第 4 种方法是通过 android.graphics.Color 类中定义好的颜色进行设置，即 layout.setBackgroundColor(Color.RED)来设置颜色，android.graphics.Color 类提供了 12 种颜色常量，如表 4.4 所示。

表 4.4　颜色常量

数 据 类 型	颜 色 常 量	常　量　值	色　码　值
int	BLACK	-16777216	0xff000000
int	BLUE	-16776961	0xff0000ff
int	CYAN	-16711681	0xff00ffff
int	DKGRAY	-12303292	0xff444444
int	GRAY	-7829368	0xff888888
int	GREEN	-16711936	0xff00ff00
int	LTGRAY	-3355444	0xffcccccc
int	MAGENTA	-65281	0xffff00ff
int	RED	-65536	0xffff0000
int	TRANSPARENT	0	0x00000000
int	WHITE	-1	0xffffffff
int	YELLOW	-256	0xffffff00

程序运行结果如图 4.14 所示，当单击了“红色”按钮后，该按钮上的文字通过 setText 方法修改为“背景红了”，窗体背景同时被修改为红色，如图 4.15 所示。



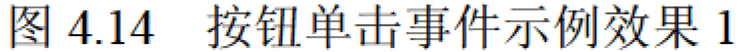


图 4.14　按钮单击事件示例效果 1



图 4.15　按钮单击事件示例效果 2

4.6　带图片的按钮——ImageButton 的使用

ImageButton 可以设计一个具有背景图片的按钮，背景图片放到 res/drawable 下，通过

ImageButton 属性 android:background="@drawable/Resource ID"设置背景图。也可以通过 ImageButton.setImageResource()实现，参数是 res/drawable 下的 Resource ID。还可以通过 selector 来设置不同状态下按钮的背景图片，这种方式需要在 res/drawable 下新增一个 XML 文件，在该 XML 中设置 state_pressed、state_checked、state_seleceted、state_focused、state_enabled 等几种状态。这个 XML 文件由<selector>标签组成，<selector>标签中可以有多个<item>标签，在<item>标签中可以定义不同状态下显示的不同图片，然后通过 ImageButton 的属性 android:background="@drawable/Resource ID"引入该<selector>标签。

ImageButton 除了可以设置背景图片外，还可以设置按钮在 OnKey、onClick、onFocusChange 等状态下显示不同的背景图片。下面代码分别对上述几种方法进行了演示，第一个 ImageButton 的背景图片通过<selector>标签来实现；第二个 ImageButton 的背景图片，通过响应不同事件在代码中实现更改背景图片；第 3 种方式是显示 Android 系统自带的图片。首先将程序中所需要的图片文件放到 res/drawable 下。ImageButtonExample .java 代码如下：

```
package com.ImageButtonExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ImageButtonExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ImageButton imgBut2=(ImageButton)this.findViewById(R.id.imgbut2);
        //获取 XML 中的 ImageButton 对象
        //在组件上按下按键时触发该事件，例如用上下方向键选择按钮
        imgBut2.setOnKeyListener(new OnKeyListener(){

         /**
          * 但在组件上按下键盘按键时触发该方法
          * @param v:触发当前事件的组件
          * @param keyCode:被按下的物理按键的编码
          * @param event:KeyEvent 对象
          * @return
          */
            @Override
            public boolean onKey(View v, int keyCode, KeyEvent event) {
                //设置 ImageButton 的背景图片，方法原型为 setImageDrawable
                (Drawable draw)
                //getResources():返回当前应用程序包下的所有资源
                ((ImageButton)v).setImageDrawable(getResources().getDrawable
                (R.drawable.pinkrose));
                setTitle(" "+keyCode);
                              //将当前按键的 keyCode 显示到标题栏中，方便测试观察

                //TODO Auto-generated method stub
                return false;
            }
        });
        //单击事件，当单击组件时触发该事件
        imgBut2.setOnClickListener(new OnClickListener(){
         /**
          * 单击组件时触发该方法
```

```
         * @param v:触发当前事件的组件
         * @return
         */
           @Override
           public void onClick(View v) {
               //TODO Auto-generated method stub
               ((ImageButton)v).setImageDrawable(getResources().getDrawable
               (R.drawable.bluerose));
           }
       });
       //焦点改变事件，当组件焦点发生变化时触发该事件
       imgBut2.setOnFocusChangeListener(new OnFocusChangeListener(){
         /**
         * 组件的焦点发生变化时触发该方法
         * @param v:触发当前事件的组件
         * @param hasFocus:当前组件的焦点状态，true：获取焦点，false：失去焦点
         * @return
         */
           @Override
           public void onFocusChange(View v, boolean hasFocus) {
               //TODO Auto-generated method stub
               if(hasFocus){
    ((ImageButton)v).setImageDrawable(getResources().getDrawable(R.draw-
    able.redrose));
                   setTitle("Focus  redrose");
               }else{
    ((ImageButton)v).setImageDrawable(getResources().getDrawable(R.draw-
    able.whiterose));
                   setTitle("Focus  whilterose");
               }
           }
       });
    }
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFEFEFEF"
    android:padding="10px">
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:layout_gravity=
        "center horizontal"
        android:text="selector 设置 ImageButton 图片" android:textColor=
        "#FF000000" />
    <ImageButton android:id="@+id/imgbut1"
        android:layout_width="wrap_content" android:layout_height="wrap_
        content"
        android:background="@drawable/buttpic" android:layout_gravity=
        "center_horizontal"
        android:layout_margin="10px" />
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:layout_gravity=
```

```
        "center horizontal"
        android:text="不同事件更改图片" android:textColor="#FF000000" />
    <ImageButton android:id="@+id/imgbut2"
        android:layout_width="wrap_content" android:layout_height="wrap_
        content"
        android:background="@drawable/whiterose" android:layout_gravity=
        "center_horizontal"
        android:layout_margin="10px" />
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:layout_gravity=
        "center_horizontal"
        android:text="调用系统默认图片" android:textColor="#FF000000" />
    <ImageButton android:id="@+id/imgbut3"
        android:layout width="wrap content" android:layout height="wrap
        content"
        android:background="@android:drawable/stat sys phone call"
        android:layout gravity="center horizontal" android:layout margin=
        "10px" />

</LinearLayout>
```

代码说明：

- id 为 imgbut1 的 ImageButton 背景图片的更改通过<selector>方式修改。android:background="@drawable/buttpic"，这里 buttpic 是在 res/drawable 下定义好的资源 buttpic.xml，该 ImageButton 按照 buttpic.xml 文件中指定的方式显示图片。
- id 为 imgbut2 的 ImageButton 通过代码实现修改背景图片。
- id 为 imgbut3 的 ImageButton 背景显示为 android:background="@android:drawable/stat_sys_phone_call"。在 Android 中可以显示用户自定义的图片，也可以显示系统自带的图片，上面的方式是显示用户自定义的图片，显示系统添加的图片格式为@android:drawable/Resource Id。系统自带的图片默认位置为 android-sdk_r06-windows\android-sdk-windows\platforms\android-3\data\res\drawable。

res/drawable/buttpic.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state pressed="false" android:drawable="@drawable/
    bluerose"/>
    <item android:state pressed="true" android:drawable="@drawable/
    redrose"/>
</selector>
```

代码说明：

<selector>用来动态改变 ImageButton 或者 ImageView 的背景，如果仅仅是单击、焦点改变等事件修改背景，通过程序代码实现很麻烦，这种情况下，可以直接通过<selector>来实现，在<selector>中可以包含多个<item>，每个<item>对应不同状态下的背景图片。android:state_pressed="false"是未单击组件状态，如果为 true，则表示单击组件，这里的第一个<item>定义了在未单击组件时显示蓝色玫瑰的图片，第二个<item>中定义了在单击组件的时候显示红色玫瑰的图片。

程序运行后效果如图 4.16 所示。当鼠标单击了第一张蓝玫瑰的图片后，效果如图 4.17 所示。

通过按键选中第二张图后的效果如图 4.18 所示。



图 4.16　ImageButton 初始状态



图 4.17　ImageButton 效果 1



图 4.18　ImageButton 效果 2

4.7　多项的选择——CheckBox 的使用

你喜欢的美食是哪一种？A：中国菜，B：日本菜，C：意大利菜，在项目中经常遇见这种让用户选择的场合，用户可以选择一项或者多项。这种场合就该 CheckBox 出马了，CheckBox 是复选框组件，有选中和非选中两种状态，可以通过 OnCheckedChangedListener() 监听该事件。下面的例子中有 4 个 CheckBox，第一个显示文字为“复选框示例”的 CheckBox 默认是选中状态；后 3 个 CheckBox，无论选中哪一个都会在 TextView 上显示你喜欢的美食信息。下面我们看一下实现过程，CheckBoxExample1 .java 代码如下：

```
package com.CheckBoxExample1;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
```

```
public class CheckBoxExample1 extends Activity {
    private TextView result;
    private String chinaStr="";
    private String jpanStr="";
    private String italianStr="";
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);          //加载 main.xml 文件
       result=(TextView)this.findViewById(R.id.result);
                            //获取 xml 文件中的定义的 TextView 组件
       CheckBox china=(CheckBox)this.findViewById(R.id.ChinaFood);
                            //获取“中国菜”CheckBox 组件
       //当 CheckBox 组件的选中和非选中状态发生变化时，触发该事件
       china.setOnCheckedChangeListener(new OnCheckedChangeListener(){
         /**
         * CheckBox 组件选中状态改变时触发该方法
         * @param buttonView:触发当前事件的组件
         * @param isChecked:当前组件的选中状态，true：选中，false：未选中
         * @return
         */
            @Override
            public void onCheckedChanged(CompoundButton buttonView,boolean
            isChecked) {
                //TODO Auto-generated method stub
                if(isChecked){
                    chinaStr=(String) buttonView.getText();
                            //获取当前 CheckBox 上显示的文字
                }else{
                    chinaStr="";
                }
                //修改 result 上面显示的文字
                result.setText("你喜欢的美食是："+chinaStr+","+jpanStr+",
                "+italianStr);
            }
       });
       CheckBox jpan=(CheckBox)this.findViewById(R.id.JapaneseFood);
       jpan.setOnCheckedChangeListener(new OnCheckedChangeListener(){
            @Override
            public void onCheckedChanged(CompoundButton buttonView,boolean
            isChecked) {
                //TODO Auto-generated method stub
                if(isChecked){
                    jpanStr=(String) buttonView.getText();
                }else{
                    jpanStr="";
                }
                result.setText("你喜欢的美食是："+chinaStr+","+jpanStr+",
                "+italianStr);
            }
       });
       CheckBox italian=(CheckBox)this.findViewById(R.id.ItalianFood);
       italian.setOnCheckedChangeListener(new OnCheckedChangeListener(){
            @Override
            public void onCheckedChanged(CompoundButton buttonView,boolean
            isChecked) {
                //TODO Auto-generated method stub
                if(isChecked){
```

```
                    italianStr=(String) buttonView.getText();
                }else{
                    italianStr="";
                }
                result.setText("你喜欢的美食是："+chinaStr+","+jpanStr+",
                "+italianStr);
            }
        });
    }
}
```

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFFFFFFF">
    <CheckBox android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="复选框示例"
        android:checked="true" android:layout_gravity="center_horizontal"
        android:textColor="#FF000000" />
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="你喜欢的美食是？"
        android:textColor="#FF000000" />
    <CheckBox android:id="@+id/ChinaFood" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="中国菜"
        android:textColor="#FF000000" />
    <CheckBox android:id="@+id/JapaneseFood" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="日本菜"
        android:textColor="#FF000000" />
    <CheckBox android:id="@+id/ItalianFood" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="意大利菜"
        android:textColor="#FF000000" />
    <TextView android:id="@+id/result" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:textColor="#FF000000"
        android:text="你喜欢的美食是:"/>
</LinearLayout>
```

代码说明：

android:checked="true"表示当前 CheckBox 为选中状态。不设置时该值默认为 false，非选中状态运行效果如图 4.19 所示。

用户也可以自定义 CheckBox，通过在<selector>中定义不同状态显示的不同图片。通过下面的例子我们看看自定义的 CheckBox 的实现过程。首先将要用到的图片保存到 res/drawable 下。

图 4.19　CheckBox 效果

CheckBoxExample2 .java 代码如下：

```
package com.CheckBoxExample2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class CheckBoxExample2 extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);       //加载 main.xml 文件
    }
}
```

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFFFFFFF">
    <CheckBox android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="自定义 CheckBox"
        android:button="@drawable/checkboxpic" android:textColor=
        "#FF000000" />
    <CheckBox android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="自定义 CheckBox
        选中状态 checked"
        android:button="@drawable/checkboxpic" android:checked="true"
        android:textColor="#FF000000" />
</LinearLayout>
```

代码说明：

android:button="@drawable/checkboxpic"，该属性用来设置按钮上的图片，即可单击部分的背景图片，例如，CheckBox、radio、button 等；而 android：background 是设置组件的背景图片，二者在效果上有很大的区别的。这里的 checkboxpic 资源位置在 res/drawable/checkboxpic.xml。在 checkboxpic.xml 中定义了 CheckBox 不同状态下显示的图片。

res/drawable/ checkboxpic.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_checked="true" android:state_focused="true"
        android:drawable="@drawable/checkbox_on_bg_focus" />
    <item android:state_checked="false" android:state_focused="true"
        android:drawable="@drawable/checkbox_off_bg_focus" />
    <item android:state_checked="true" android:drawable="@drawable/
    checkbox_on_bg" />
    <item android:state_checked="false" android:drawable="@drawable/
    checkbox_off_bg" />
</selector>
```

代码说明：

- android:state_checked="true"表示组件选中状态，值为 false 表示非选中状态；android:state_focused="true"表示组件获取焦点状态，值为 false 表示失去焦点状态。
- 第 1 个<item>标签定义了当组件被选中，并获取焦点时显示的图片。
- 第 2 个<item>标签定义了当组件未被选中，但获取焦点时显示的图片。
- 第 3 个<item>标签定义了当组件选中时的背景图片。
- 第 4 个<item>标签定义了当组件未被选中时的背景图片。

注意<item>的前后顺序，获取焦点时，选中和未选中状态的<item>要放到前面。



图 4.20　自定义 CheckBox

运行效果如图 4.20 所示。

4.8　唯一的性别——RadioButton 和 RadioGroup 的使用

RadioButton 是单选按钮组件，RadioGroup 可以将不同的 RadioButton 限制在一个按钮组里。在一个按钮组里的 RadioButton，只能有一个处于选中状态，可以通过 setOnCheckedChangeListener 去监听按钮组上 RadioButton 的状态变化。

下面的例子中添加 3 个 RadioButton，将 3 个 RadioButton 放到 RadioGroup 中，在单击 RadioButton 时，显示当前选择的是哪一种水果。单击“清除”按钮可以清除 RadioGroup 中 RadioButton 的选中状态，这里的 RadioButton 使用自定义图片显示，通过<selector>标签实现，方法和 CheckBox 实现相同。将项目中用到的图片和<selector>的 XML 文件放到 res/drawable 下。

RadioButtonExample .java 代码如下：

```
package com.RadioButtonExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class RadioButtonExample extends Activity {
    private TextView result;
    private RadioGroup radioGroup;
    private RadioButton rb1;
    private RadioButton rb2;
    private RadioButton rb3;
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);              //加载 main.xml 资源文件
       result=(TextView)this.findViewById(R.id.result);//获取 result 组件
       rb1=(RadioButton)this.findViewById(R.id.RB1);   //获取 RB1 组件
       rb2=(RadioButton)this.findViewById(R.id.RB2);   //获取 RB2 组件
       rb3=(RadioButton)this.findViewById(R.id.RB3);   //获取 RB3 组件
       radioGroup=(RadioGroup)this.findViewById(R.id.radioGroup);
                                                   //获取 RadioGroup 组件
       //当 RadioGroup 中的 RadioButton 状态改变时，执行该事件
       radioGroup.setOnCheckedChangeListener(new OnCheckedChangeListener(){
            /*
             * RadioButton 状态改变时调用该方法
             * group：触发当前事件的 RadioGroup
             * checkedId：触发当前事件的 RadioButton 的 id。
             */
            @Override
            public void onCheckedChanged(RadioGroup group, int checkedId) {
                //TODO Auto-generated method stub
                if(checkedId==rb1.getId()){
                                       //判断当前触发事件的组件是否是 rb1 组件
                    result.setText("您选择的是："+rb1.getText());
                                       //将 rb1 上的文字显示在 result 上
                }
                if(checkedId==rb2.getId()){
                    result.setText("您选择的是："+rb2.getText());
                }
                if(checkedId==rb3.getId()){
```

```
                        result.setText("您选择的是："+rb3.getText());
                    }
                }
            });
            Button clear=(Button)this.findViewById(R.id.clear);//获取 clear 组件
            clear.setOnClickListener(new OnClickListener(){     //组件单击事件
                @Override
                public void onClick(View v) {
                    //TODO Auto-generated method stub
                    radioGroup.clearCheck();        //清除 RadioGroup 上组件状态
                    result.setText("");
                }
            });
        }
}
```

用于设置 RadioButton 背景的 XML 文件为 res/drawable/radiobuttonpic.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 组件选中时，加载图 radiobutton_on_bg -->
    <item android:state_checked="true" android:drawable="@drawable/
    radiobutton_on_bg" />
    <!-- 组件非选中时，加载图 radiobutton_off_bg -->
    <item android:state_checked="false" android:drawable="@drawable/
    radiobutton_off_bg" />
</selector>
```

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFFFFFFF"
    android:padding="10px">
    <RadioGroup android:id="@+id/radioGroup"
        android:layout_width="wrap_content" android:layout_height="wrap_
        content">
        <RadioButton android:id="@+id/RB1" android:layout_width="wrap_
        content"
            android:layout_height="wrap_content" android:text="水蜜桃"
            android:textColor="#FF000000" android:button="@drawable/
            radiobuttonpic"/>
        <RadioButton android:id="@+id/RB2" android:layout_width="wrap_
        content"
            android:layout_height="wrap_content" android:text="苹果"
            android:textColor="#FF000000" android:button="@drawable/
            radiobuttonpic"/>
        <RadioButton android:id="@+id/RB3" android:layout_width="wrap_
        content"
            android:layout_height="wrap_content" android:text="荔枝"
            android:textColor="#FF000000" android:button="@drawable/
            radiobuttonpic"/>
    </RadioGroup>
    <TextView android:id="@+id/result" android:layout_width="wrap_
    content"
        android:layout_height="wrap_content" android:textColor="#FF000000" />
    <Button android:id="@+id/clear" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="清除" />
</LinearLayout>
```

代码说明如下。

- <RadioButton>：单选按钮标签，可以设置默认是否选中，通过属性 android:checked 设置，如果该值为 true，则未选中状态，否则为非选中状态。
- <RadioGroup>：用于限制多个 RadioButton，只能有一个为选中状态。

运行效果如图 4.21 所示。

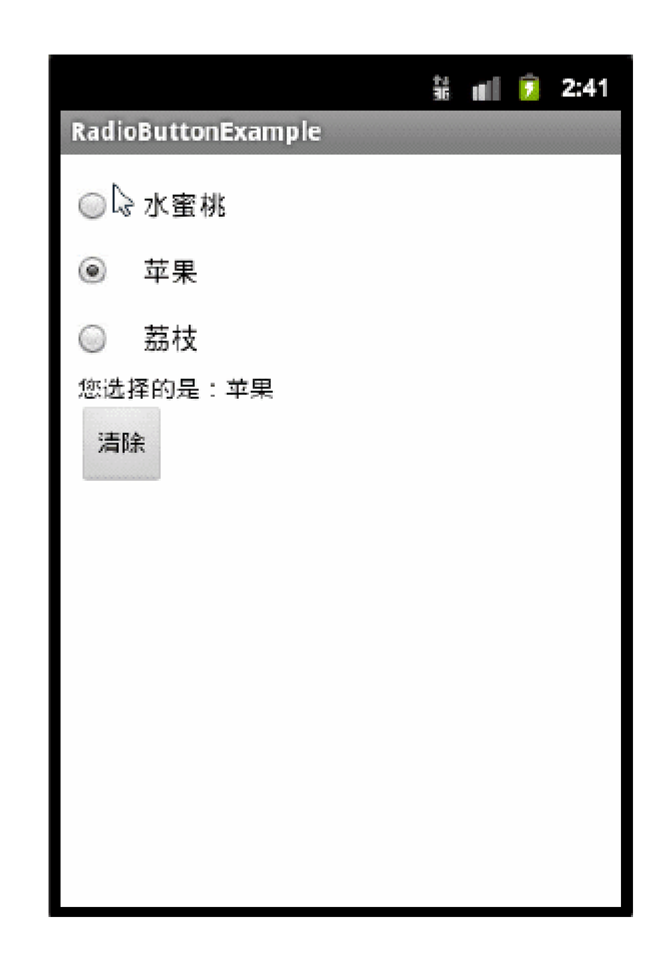


图 4.21　RadioButton 和 RadioGroup 示例

4.9　请稍等的提示——ProgressDialog 的使用

经常会看见一些应用程序，在请求数据的时候会出现“加载中”对话框，Android 中 ProgressDialog 类用来实现该效果。ProgressDialog 对话框可以设置对话框上显示的文字、图标、进度条的样式。ProgressDialog 中常用方法如表 4.5 所示。

表 4.5　ProgressDialog常用方法说明

方法原型	功　　能	参数说明
public void setProgressStyle (int style)	设置进度条的样式	style：进度条样式，在 ProgressDialog 定义了两种样式常量，一种是 STYLE_HORIZONTAL（长型进度条），另一种是 STYLE_SPINNER（圆形进度条）
public void setTitle (CharSequence title)	设置进度条的标题	title：标题
public void setMessage (CharSequence message)	设置提示信息文字	message：提示信息
public void setIcon (int resId)	设置图标	resId：图片常量值，例如 R.drawable.pic
public void setIndeterminate (boolean indeterminate)	设置进度条是否不明确	true：设置为不明确。false：不设置为不明确
public void setCancelable (boolean flag)	是否可以按退回键取消	true：可以，false：不可以

续表

方 法 原 型	功　　能	参 数 说 明
public void setButton (CharSequence text, DialogInterface. OnClickListener listener)	设置进度条上的按钮文字信息及事件	text：按钮上的文字信息，listener：按钮单击事件
public void show ()	显示进度条	
public void setMax (int max)	设置进度条的最大值	max：最大的取值
public void setProgress (int value)	设置进度条当前进度	value：当前进度值
public void setSecondaryProgress (int secondaryProgress)	设置第二进度条当前值	value：第二进度条当前值
public void dismiss ()	释放对话框，从当前窗体移除	

下面我们在一个 Activity 中添加两个 Button，分别定义了圆形进度条和长形进度条。

ProgressDialogExample .java 代码如下：

```
package com.ProgressDialogExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ProgressDialogExample extends Activity {
    Button circleBut;
    Button longBut;
    ProgressDialog myDialog;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); //加载 main.xml
        circleBut = (Button) this.findViewById(R.id.circleButt);
                                //获取 circleButt“圆形进度条”按钮
        longBut = (Button) this.findViewById(R.id.longButt);
                                //获取 longButt“长形进度条”按钮
        circleBut.setOnClickListener(new OnClickListener() {
                                //为“圆形进度条”添加单击事件
                @Override
                public void onClick(View v) {
                    //TODO Auto-generated method stub
                    myDialog = new ProgressDialog(
                            ProgressDialogExample.this);
                                //创建 ProgressDialog 对象
                    //设置进度条的样式为圆形样式
                    myDialog.setProgressStyle(ProgressDialog.STYLE_
                    SPINNER);
                    myDialog.setTitle("提示");
                                //设置进度条的标题信息
                    myDialog.setMessage("数据加载中,请稍后...");
                                //设置进度条的提示信息
                    myDialog.setIcon(R.drawable.android);
                                //是指进度条的图标
                    myDialog.setIndeterminate(false);
                                //设置进度条是否为不明确
```

```
                    myDialog.setCancelable(true);
                                //设置进度条是否按返回键取消
                    //为进度条添加“确定”按钮，并为该按钮添加单击事件
                    myDialog.setButton("确定",
                            new DialogInterface.OnClickListener() {
                                @Override
                                public void onClick(DialogInterface
                                dialog,int which) {
                                    //TODO Auto-generated method stub
                                    myDialog.cancel();  //撤销进度条
                                }
                            });
                    myDialog.show();                    //显示进度条
                }
            });
        longBut.setOnClickListener(new OnClickListener() {
                                        //为“长形进度条”添加单击事件
                int count = 0;          //存储进度条当前进度值，初始值为 0
                @Override
                public void onClick(View v) {
                    //TODO Auto-generated method stub
                    myDialog = new ProgressDialog(
                            ProgressDialogExample.this);
                    myDialog
                            .setProgressStyle(ProgressDialog.STYLE
                            HORIZONTAL);
                    myDialog.setTitle("提示");
                    myDialog.setMessage("数据加载中,请稍后...");
                    myDialog.setIcon(R.drawable.android);
                    myDialog.setIndeterminate(false);
                                    //设置进度条是否为不明确
                    myDialog.setCancelable(true);
                    myDialog.setMax(200);   //设置进度条的进度最大值
                    myDialog.setProgress(0);//设置当前默认进度为 0
                    myDialog.setSecondaryProgress(100);
                                    //设置第二进度条的值为 100
                    //为进度条添加“取消”按钮，并为该按钮添加单击事件
                    myDialog.setButton("取消",
                            new DialogInterface.OnClickListener() {
                                @Override
                                public void onClick(DialogInterface
                                dialog,int which) {
                                    //TODO Auto-generated method stub
                                    myDialog.cancel();
                                }
                            });
                    myDialog.show();    //显示进度条
                    new Thread() {      //定义线程，动态改变当前进度条的值
                        public void run() {
                            while (count <= 200) {
                                myDialog.setProgress(count++);
                                        //设置当前进度条的进度值
                                try {
                                    Thread.sleep(100);  //暂停 0.1 秒
                                } catch (InterruptedException e) {
                                    //TODO Auto-generated catch block
                                    e.printStackTrace();
                                }
```

```
                            }
                        }
                    }.start();                    //启动线程
                }
            });
    }
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFFFFFFF"
    android:padding="10px">
    <Button android:id="@+id/circleButt" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="圆形进度条"
        android:layout_weight="1" />
    <Button android:id="@+id/longButt" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="长形进度条"
        android:layout_weight="1" />
</LinearLayout>
```

单击“圆形进度条”程序运行效果如图 4.22 所示。单击“长形进度条”效果如图 4.23 所示。



图 4.22　圆形进度条效果



图 4.23　长形进度条

4.10　后台程序完成读数据——ProgressBar 与 Handler

ProgressBar 和 ProgressDialog 一样，用来显示进度条。ProgressDialog 是进度条对话框，运行时以对话框的形式呈现，此时应用程序将失去焦点，直到进程结束，才将主动权交给应用程序，需要在代码中创建 ProgressDialog 对象使用。ProgressBar 是进度条组件，可以在 XML 中定义，为用户呈现操作的进度，和 ProgressDialog 类似，还有一个次要进度条，用来显示中间进度，如在播放流媒体时缓冲的进度。进度条在不确定模式下，显示循环动

画，一般在应用程序使用任务长度未知的时候使用。

进度条组件一般和 Handler 配合使用。Handler 主要用来接受子线程发送的数据，配合主线程更新 UI。下面我们通过例子，了解一下 Handler 的使用。首先思考一个问题，我们如何实现 6 秒钟更新 TextView 内容。很多习惯了写 Java 程序的朋友经常会用下面这种方式实现。

```
package com. HandlerExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class HandlerExample extends Activity {
    private TextView myTextView;
    private int count = 0;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);     //加载布局资源文件 main.xml
        myTextView=(TextView) this.findViewById(R.id.mess);
                                           //获取布局文件中的 TextView 组件
        Timer timer = new Timer();
        timer.scheduleAtFixedRate(new MyShedule(), 1, 6000);
                                           //每隔 6 秒钟执行 MyShedule
    }
    private class MyShedule extends TimerTask {
        @Override
        public void run() {
            myTextView.setText(new Integer(count).toString());
                                           //设置文本框上显示的进度值
            count++;
        }
    }
}
```

但这种实现方式却不能实现我们想要达到的效果。所以在 Android 中出现 Handler 这个类，它是 Runnable 和 Activity 交互的桥梁。在 run()方法中发送 Message，在 Handler 里，通过不同的 Message 执行不同的任务。对上面的代码修改为如下内容，便可实现预期效果。

```
package com.HandlerExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class HandlerExample extends Activity {
    private TextView myTextView;
    private int count = 0;
    protected static final int Start_NOTIFIER = 0x101;
                                    //自定义 Handler 信息代码，用以作为识别事件处理
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);         //加载 main.xml 布局文件
        myTextView=(TextView) this.findViewById(R.id.mess);
                                               //获取 TextView 对象
        Timer timer = new Timer();             //创建 Timer 时钟对象
        timer.scheduleAtFixedRate(new MyShedule(), 1, 6000);
                                               //每隔 6 秒钟执行 MyShedule
    }
    private class MyShedule extends TimerTask {
        @Override
```

```
        public void run() {
            Message message=new Message();      //创建 Message 对象
            message.what=HandlerExample.Start_NOTIFIER;
                                    //用户自定义消息代码，以便收件人找到讯息是什么
            handler.sendMessage(message);       //向 Handler 发送消息
        }
    }
    //创建 Handler 对象，通过实现 handleMessage 方法，接收信息
    Handler handler=new Handler(){
        public void handleMessage(Message msg) {
                                    //子类必须实现该方法才可以接收到信息
            switch(msg.what){   //判断消息代码值
            case HandlerExample.Start_NOTIFIER:
                myTextView.setText(new Integer(count).toString());
                                    //修改 TextView 显示的文字
                count++;
                break;
            }
        }
    };
}
```

下面实现一个 Handler 和 ProgressBar 配合使用的例子，在窗体上添加一个 ProgressBar 组件、一个 TextView 组件和一个 Button 组件，单击 Button，修改 ProgressBar 的当前进度值，并在 TextView 中显示当前进度值。通过 count 对当前进度值进行控制。

ProgressBarExample .java 代码如下：

```
package com.ProgressBarExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ProgressBarExample extends Activity  {
    private TextView myTextView;
    private Button myButton;
    private ProgressBar myprogressBar;
    public int intCounter = 0;

    //自定义 Handler 信息代码，用以作为识别事件处理
    protected static final int STOP_Flag = 0x100;
    protected static final int THREADING_Flag = 0x101;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);//加载 main.xml 文件
        myButton = (Button) findViewById(R.id.download);
                                        //获取名字为 download 的 Button 组件
        myTextView = (TextView) findViewById(R.id.showmes);
                                        //获取名字为 showmes 的 TextView 组件
        myprogressBar = (ProgressBar) findViewById(R.id.progressbar);
                                        //获取名字为 pb 的 ProgressBar 组件
        myButton.setOnClickListener(new OnClickListener(){
                                        //按钮的单击事件
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                myTextView.setText(R.string.start);//设置按钮上显示的文字
                myprogressBar.setVisibility(View.VISIBLE);
                                        //设置 myprogressBar 为可见状态
```

```
            myprogressBar.setMax(100);
                                    //设置 myprogressBar 进度的最大值为 100
            myprogressBar.setProgress(0);  //设置当前值为 0
            myprogressBar.setIndeterminate(false);
                                    //设置进度条为明确显示
            new Thread(new Runnable() {
                public void run() {
                    while(intCounter<=100){
                        //计数器值小于等于 100 的时候，修改进度条当前进度值
                        try {
                            intCounter = intCounter+1;  //计数器累加 1
                            Thread.sleep(100);          //进程休眠 0.1 秒
                          //如果计数器 intCounter 累加到 100，则向 Handler
                          发送 STOP Flag 消息
                            if (intCounter ==100) {
                                Message m = new Message();
                                                    //定义 Message 对象
                                //自定义 Message 的消息信息代码
                                m.what = ProgressBarExample.STOP Flag;
                                //向 Handler 发送消息信息代码 ProgressBar-
                                Example.this.myMessageHandler.sendMe-
                                ssage(m);break;
                            //如果计数器不等于 100，则向 Handler 发送
                            THREADING Flag 消息
                            } else {
                                Message m = new Message();
                                m.what = ProgressBarExample.THREADING_
                                Flag;
                                ProgressBarExample.this.myMessageHandler
                                .sendMessage(m);
                            }
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }
                }
            }).start();
        }

    });
}

Handler myMessageHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
        case ProgressBarExample.STOP Flag:
                                    //如果消息代码为 STOP_Flag，表示下载完毕
            myTextView.setText("下载完毕");//修改 myTextView 上显示文字
            myButton.setText("下载完毕");  //修改 myButton 上显示文字
            myprogressBar.setVisibility(View.GONE);
                                    //设置 myprogressBar 为不可见
            Thread.currentThread().interrupt();    //中断当前线程
            break;
        //如果消息代码为 THREADING_Flag，表示正在下载中
        case ProgressBarExample.THREADING_Flag:myprogressBar.
        setProgress(intCounter);    //修改 myprogressBar 的当前进度值
                //修改 myTextView 上显示文字
```

```
                    myTextView
                            .setText(getResources().getText(R.string.start)
                + "("
                                    + Integer.toString(intCounter)
                                    + "%)\n"
                                    + "Progress:"
                                    + Integer.toString(myprogressBar
                                            .getProgress())
                                    );
                break;
            }
            super.handleMessage(msg);
        }
    };
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFFFFFFF" >
    <!-- 定义 ProgressBar 组件
    style="?android:attr/progressBarStyleHorizontal": 设置进度条为水平进度条
     -->
    <ProgressBar android:id="@+id/progressbar"
        android:layout width="fill parent" android:layout height="wrap
        content"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_margin="10px" />
    <Button android:id="@+id/download" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="下载" />
    <TextView android:id="@+id/showmes" android:layout width="fill parent"
        android:layout height="wrap content" android:textColor="#FF000000" />
</LinearLayout>
```

Res/values/strings.xml 内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="start">开始下载....</string>
   <string name="app_name">ProgressBarExample</string>
</resources>
```

当单击“下载”按钮后，运行结果如图 4.24 所示。

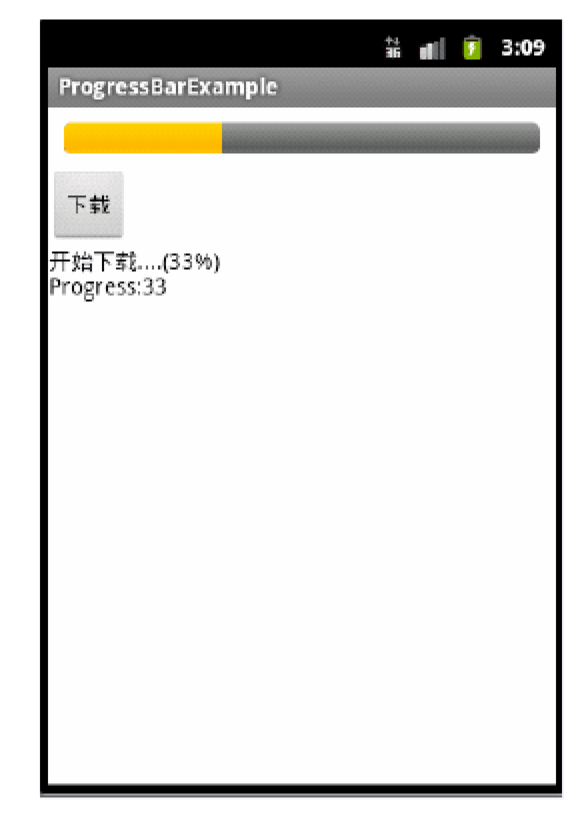


图 4.24　ProgressBar 和 Handler

4.11　设置日期——DatePickerDialog 的使用

DatePickerDialog 是设置日期对话框，通过 OnDateSetListener 监听重新设置日期事件，当日期被设置后，会执行 OnDateSetListener 类中的方法 onDateSet。下面我们在窗体上添加一个 Button，单击该 Button 出现设置系统日期对话框，修改日期后，在窗体的 TextView 中显示新的日期。

DatePickerDialogExample .java 代码如下：

```
package com.DatePickerDialogExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class DatePickerDialogExample extends Activity {
    private TextView showDate;
    private Button setDate;
    private int year;
    private int month;
    private int day;
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main); //获取 main.xml 文件
       showDate=(TextView) this.findViewById(R.id.showDate);
                                    //获取用来显示当前日期的 TextView 组件
       setDate=(Button) this.findViewById(R.id.setDate);//获取 Button 组件
       //初始化 Calendar 日历对象。
       Calendar myCalendar = Calendar.getInstance(Locale.CHINA);
       Date myDate=new Date();          //获取当前日期 Date 对象
       myCalendar.setTime(myDate);      //为 Calendar 对象设置时间为当前日期
       year=myCalendar.get(Calendar.YEAR);      //获取 Calendar 对象中的年
       month=myCalendar.get(Calendar.MONTH);
                              //获取 Calendar 对象中的月，0 表示 1 月，1 表示 2 月……
       day=myCalendar.get(Calendar.DAY_OF_MONTH);  //获取这个月的第几天
       showDate.setText(year+"-"+(month+1)+"-"+day);
                              //修改 TextView 显示的信息为当前的年月日
       setDate.setOnClickListener(new OnClickListener(){
                              //“设置日期”按钮的单击事件
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                   //创建 DatePickerDialog 对象。
        //构造函数原型:public DatePickerDialog (Context context, DatePickerDialog.
        OnDateSetListener callBack, int year, int monthOfYear, int dayOfMonth)
        //参数含义依次为 context：组件运行 Activity,DatePickerDialog.OnDate-
        SetListener:选择日期事件
        //year:当前组件上显示的年，monthOfYear：当前组件上显示的月，dayOfMonth：
        当前组件上显示的日
                   DatePickerDialog dpd=new DatePickerDialog(DatePicker-
                   DialogExample.this,new OnDateSetListener(){
                       /*
                        * view:该事件关联的组件
                        * myyear：当前选择的年
```

```
                                        * monthOfYear:当前选择的月
                                        * dayOfMonth: 当前选择的日
                                        */
                                       @Override
                                       public void onDateSet(DatePicker view, int myyear,
                                       int monthOfYear,int dayOfMonth) {
                                           //在 DatePickerDialog 组件上设置日期后，同时修改
                                           TextView 上的信息
                                           showDate.setText(myyear+"-"+(monthOfYear+1)+
                                           "-"+dayOfMonth);
                                           //修改 year，month，day 变量值，以便在依次单击按钮时
                                           DatePickerDialog 上显示上一次修改后的值
                                            year=myyear;
                                            month=monthOfYear;
                                            day=dayOfMonth;

                                       }},year,month,day);
                                  dpd.show();//显示 DatePickerDialog 组件
                      }
                 });
            }
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout height="fill parent" android:background="#FFFFFFFF">
    <TextView android:id="@+id/showDate" android:layout width="wrap content"
        android:layout height="wrap content" android:textColor="#FF000000" />
    <Button android:id="@+id/setDate" android:layout width="wrap content"
        android:layout_height="wrap_content" android:text="设置日期" />
</LinearLayout>
```

运行结果如图 4.25 所示。



图 4.25　单击“设置日期”按钮效果

4.12　动态输入日期和时间——TimePickerDialog 的使用

TimePickerDialog 是设置时间对话框，通过 OnTimeSetListener 监听重新设置时间事件。下面在窗体中添加一个 TextView 用来显示当前时间，单击 Button 组件弹出设置时间对话框。时间可以设置成 24 小时制和 12 小时制，如果是 12 小时制，弹出框上会有一个“AM”和“PM”切换按钮，用来选择是上午还是下午。下面我们一起看下面的例子。

TimePickerDialogExample .java 代码如下：

```
package com.TimePickerDialogExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class TimePickerDialogExample extends Activity {
    private TextView showTime;
    private Button setTime;
    private int year;
    private int month;
    private int day;
    private int hour;
    private int minus;
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);  /加载 main.xml 布局文件文件
       showTime=(TextView) this.findViewById(R.id.showTime);
                                       //获取用来显示当前时间的 TextView 组件
       setTime=(Button) this.findViewById(R.id.setTime);
                                       //获取设置时间 Button 组件
       Calendar myCalendar = Calendar.getInstance(Locale.CHINA);
                                       //初始化 Calendar 日历对象。
       Date myDate=new Date();         //获取当前日期 Date 对象
       myCalendar.setTime(myDate);     //为 Calendar 对象设置时间为当前日期
       year=myCalendar.get(Calendar.YEAR); //获取 Calendar 对象中的年
       month=myCalendar.get(Calendar.MONTH);
                        //获取 Calendar 对象中的月，0 表示 1 月，1 表示 2 月……
       day=myCalendar.get(Calendar.DAY_OF_MONTH);      //获取这个月的第几天
       hour=myCalendar.get(Calendar.HOUR_OF_DAY);      //获取小时信息
       minus=myCalendar.get(Calendar.MINUTE);          //获取分钟信息
       //设置 TextView 组件上显示的日期信息
       showTime.setText(year+"-"+(month+1)+"-"+day+""+hour+":"+minus);
       setTime.setOnClickListener(new OnClickListener(){
                                              // “设置日期”按钮的单击事件
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
        //创建 TimePickerDialog 对象
        //构造函数原型：TimePickerDialog(Context context, TimePickerDialog.
        OnTimeSetListener callBack, int hourOfDay, int minute, boolean
        is24HourView)
       //参数含义依次为 context:组件运行 Activity,TimePickerDialog.OnTimeSet-
       Listener:选择时间事件
```

```
        //hourOfDay:当前组件上显示小时，minute：当前组件上显示的分钟，is24HourView：
        是否是 24 小时方式显示，或者 AM/PM 方式显示
                TimePickerDialog tpd=new TimePickerDialog(TimePicker
                DialogExample.this,new TimePickerDialog.OnTimeSetListener(){
                    @Override
                    public void onTimeSet(TimePicker view, int hourOfDay,
                            int myminute) {
                        //TODO Auto-generated method stub
                        showTime.setText(year+"-"+(month+1)+"-"+day+"
                        "+hourOfDay+":"+myminute);
                        hour=hourOfDay;
                       minus=myminute;
                    }

                },hour,minus,false);
                tpd.show();
            }
        });
    }
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFFFFFFF">
    <TextView android:id="@+id/showTime" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:textColor="#FF000000" />
    <Button android:id="@+id/setTime" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="设置日期" />
</LinearLayout>
```

运行结果如图 4.26 所示。



图 4.26　TimePickerDialog 示例

4.13　提示信息——Toast 的使用

Toast 是 Android 中用来显示提示信息的一种机制，和 Dialog 不同，它没有焦点。Toast 显示的时间有限，过一定时间后自动消失。Toast 可以自定义提示框的位置、显示文字、显示的 ICON 等信息。

ToastExample .java 代码如下：

```
package com.ToastExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ToastExample extends Activity {
     Button defaultToast;
     Button defineToast;
     Button iconToast;
     Button defineAllToast;
     Toast toast;
     /** Called when the activity is first created. */
     @Override
     public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);  //加载布局资源文件
        defaultToast = (Button) this.findViewById(R.id.defaultToast);
                                                   //加载布局文件中的默认 Toast 按钮
        defaultToast.setOnClickListener(new OnClickListener() {
                                                   //监听 defaultToast 单击事件
            @Override
            public void onClick(View v) {
                 // makeText 方法 3 个参数，第一个参数：Context;第二个参数：提示信息；
                 第三个参数：信息框消失方式，有两种取值 Toast.LENGTH_SHORT（在
                 短时间内消失）和 Toast.LENGTH_LONG(较长时间消失)
                Toast.makeText(ToastExample.this, R.string.ToastText,
                        Toast.LENGTH_SHORT).show();
            }
        });
        //自定义显示位置 Toast
        defineToast = (Button) this.findViewById(R.id.defineToast);
        defineToast.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                toast = Toast.makeText(ToastExample.this, R.string.ToastText,
                        Toast.LENGTH_SHORT);
                //提示框出现的位置，参数 1：位置通过 Gravity 类设置，参数 2：x 偏移量，
                参数 3：y 偏移量
                toast.setGravity(Gravity.CENTER, 0, 0);
                toast.show();                              //显示 Toast
            }
        });
        //带图标的 Toast
        iconToast = (Button) this.findViewById(R.id.IconToast);
        iconToast.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
                //TODO Auto-generated method stub
                toast = Toast.makeText(ToastExample.this, R.string.ToastText,
                        Toast.LENGTH SHORT);
                toast.setGravity(Gravity.CENTER, 0, 0);
                LinearLayout view = (LinearLayout) toast.getView();
                                        //getView():获取 Toast 的 View 对象
                ImageView imgView = new ImageView(ToastExample.this);
                                        //创建 ImageView 对象
                imgView.setImageResource(R.drawable.icon);
                                        //设置 imgView 的背景图片
                view.addView(imgView);  //将 imgView 添加到 View 上
                toast.setView(view);    //将 view 显示在 Toast 上
                toast.show();           //显示 Toast
            }
        });
        //完全自定义 Toast
        defineAllToast = (Button) this.findViewById(R.id.defineAllToast);
        defineAllToast.setOnClickListener(new OnClickListener() {
             @Override
             public void onClick(View v) {
                 //TODO Auto-generated method stub
                 toast = new Toast(ToastExample.this);
                 LayoutInflater inflater = getLayoutInflater();
                                        //获取 LayoutInflater 对象
        //inflate():将 Layout 文件转换为 View，这里是将 definetoast.xml 中的
        myToastLayout 组件转化为 View
                View myToastLayout = inflater.inflate(R.layout.definetoast,
                        (ViewGroup) findViewById(R.id.myToastLayout));
                toast.setGravity(Gravity.RIGHT | Gravity.BOTTOM, 40, 40);
                //设置提示信息出现的位置
                toast.setDuration(Toast.LENGTH LONG);
                                        //设置如何显示提示信息
                toast.setView(myToastLayout);
                                        //将 myToastLayout 显示在 Toast 上
                toast.show();           //显示 Toast
            }
        });
    }
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout height="fill parent">
    <Button android:id="@+id/defaultToast" android:layout_width="wrap_
    content"
        android:layout_height="wrap_content" android:text="默认 Toast" />
    <Button android:id="@+id/defineToast" android:layout_width="wrap_
    content"
        android:layout_height="wrap_content" android:text="自定义位置 Toast" />
    <Button android:id="@+id/IconToast" android:layout width="wrap ontent"
        android:layout_height="wrap_content" android:text="带图标 Toast" />
    <Button android:id="@+id/defineAllToast" android:layout width="wrap
    ontent"
```

```
        android:layout_height="wrap_content" android:text="完全自定义 Toast" />
</LinearLayout>
```

Res/layout/definetoast.xml 是完全自定义提示框的布局文件，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="wrap_content" android:layout_width="wrap_content"
    android:background="#ffffffff" android:orientation="vertical"
    android:id="@+id/myToastLayout">
    <TextView android:layout_height="wrap_content"
        android:layout_margin="1dip" android:textColor="#ffffffff"
        android:layout_width="fill_parent" android:gravity="center"
        android:background="#cc000000" android:id="@+id/tvTitleToast"
        android:text="提示信息" />
    <LinearLayout android:layout_height="wrap_content" android:layout_
    width="wrap_content"
    android:background="#FFE0E0E0" android:orientation="vertical">
        <ImageView android:layout_height="wrap_content"
            android:layout_gravity="center" android:layout_width="wrap_
            content"
            android:background="@drawable/icon" />
        <TextView android:layout_height="wrap_content"
            android:paddingRight="10dip" android:paddingLeft="10dip"
            android:layout_width="wrap_content" android:gravity="center"
            android:textColor="#ff000000"
            android:text="完全自定义 Toast" />
    </LinearLayout>
</LinearLayout>
```

单击“默认 Toast”按钮，效果如图 4.27 所示。单击“自定义位置 Toast”按钮，效果如图 4.28 所示。



图 4.27　默认 Toast



图 4.28　自定义位置 Toast

单击“带图标 Toast”按钮，效果如图 4.29 所示。单击“完全自定义 Toast”按钮，效果如图 4.30 所示。




图 4.29　带图标 Toast



图 4.30　完全自定义 Toast

4.14　自定义下拉菜单——Spinner

Spinner 是下拉菜单组件，类似于 HTML 中的<select>，每次只能选择所有项目中的一项。Spinner 上的选项来自于与之关联的适配器（ArrayAdapter）中，下拉菜单的样式通过 setDropDownViewResource()方法定义，参数为 XML 资源。Spinner 通过 getSelectedItem()方法获取当前选中项的信息。下面在窗体上添加一个 Spinner，用来显示城市，通过 setOnItemSelectedListener 事件监听选择下拉菜单中某一项，显示该选中项的下标和 id 及当前选择的城市。

SpinnerExample .java 代码如下：

```
package com.SpinnerExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class SpinnerExample extends Activity {
    private Spinner mySpinner;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);// 加载 main.xml 文件
        //第 1 步：获取 XML 中的 Spinner 组件
        mySpinner = (Spinner) this.findViewById(R.id.mySpinner);
        //第 2 步：为下拉列表项定义适配器
        //createFromResource (Context context, int textArrayResId, int
        //textViewResId) 参数的含义为
        //1.context:应用上下文
        //2.textArrayResId：适配器的数据源，这里的 R.array.colors 是在
        res/values/arrays.xml 中定义的数组
        //3.Spinner 上显示数据的视图。这里用 Android 自带的简单的下拉菜单方式
```

```
        ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
                this, R.array.colors, android.R.layout.simple_spinner_item);
        //第 3 步：设置当 Spinner 按下时在下拉列表里显示数据视图
        adapter.setDropDownViewResource(android.R.layout.simple_spinner
        dropdown_item);
        //第 4 步：为 Spinner 添加适配器
        mySpinner.setAdapter(adapter);
        //第 5 步：为 Spinner 添加事件监听，setOnItemSelectedListener 该事件在菜
        单被选中时触发
        mySpinner.setOnItemSelectedListener(new OnItemSelectedListener() {
            /*
            * 功能：Spinner 的项被选择时触发该方法 parent 当前被选择的对象所在的
            AdapterView view
            * 在 AdapterView 中被单击的 View position 当前单击项在 View 的位置，
            从 0 开始，0 是第一项 id
            * 被选择项的 id
            */
            @Override
            public void onItemSelected(AdapterView<?> parent, View view,
                    int position, long id) {
                //TODO Auto-generated method stub
                //将当前单击项的坐标和 id 显示出来
                //getSelectedItem():获取选中项的值
                Toast.makeText(
                        SpinnerExample.this,
                        "position:" + position + " id:" + id + " value:"
                                + mySpinner.getSelectedItem().toString(),
                        Toast.LENGTH_SHORT).show();
            }

            @Override
            public void onNothingSelected(AdapterView<?> arg0) {
                // TODO Auto-generated method stub
                Toast.makeText(SpinnerExample.this, "unselected",
                        Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

Res/values/arrays.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- 自定义数组，通过 name 调用该数组，数组中有几个元素，就用几个 item 标签-->
   <string-array name="colors">
      <item>北京</item>
      <item>上海</item>
      <item>天津</item>
      <item>深圳</item>
   </string-array>
</resources>
```

Res/values/strings.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello">Hello World, Spinner!</string>
   <string name="app_name">SpinnerExample</string>
   <string name="spinner_title">请选择城市</string>
</resources>
```

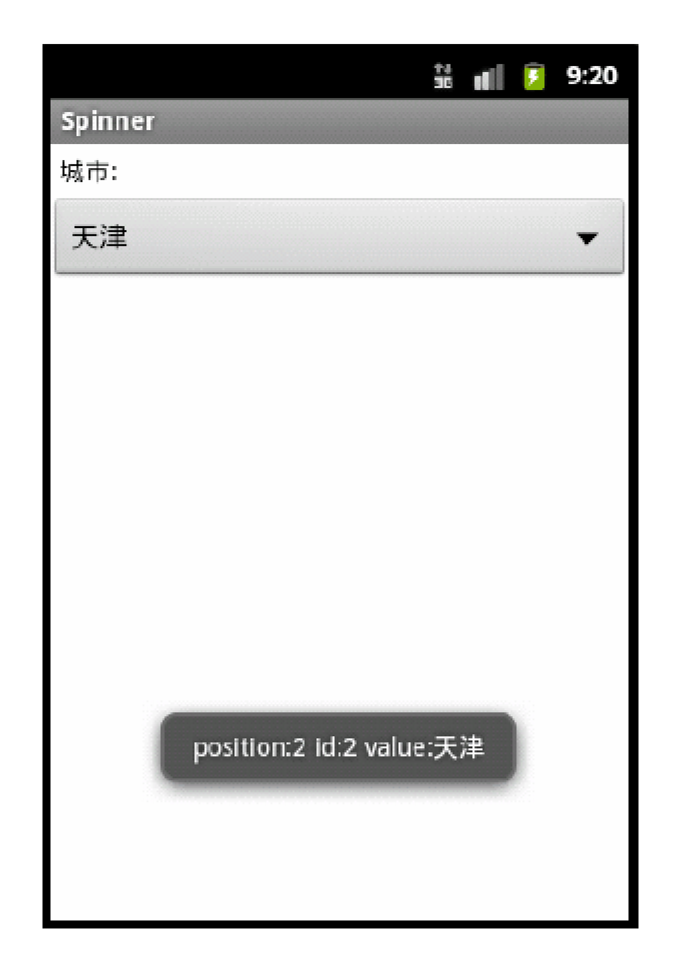
Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#ffffffff">
    <TextView android:layout_width="fill_parent" android:layout_height=
    "wrap_content"
    android:text="城市:" android:textColor="#ff000000" android:layout_
    margin="5px"/>
    <!-- Spinner:下拉菜单组件 。 android:prompt: 下拉菜单中的标题-->
    <Spinner android:id="@+id/mySpinner" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:prompt="@string/sp-
        inner_title"
        />
</LinearLayout>
```

运行效果如图 4.31 和图 4.32 所示。



图 4.31　单击 Spinner 后效果图



4.32　单击 Spinner 提示效果图

4.15　动态添加/删除下拉菜单——Spinner

通过上一节的学习，我们已经对 Spinner 的事件处理有所了解。有时候项目中涉及要动态地更新 Spinner 菜单中的内容，这时候就需要通过 ArrayList 的依赖性来完成。下面我们先认识一下 ArrayAdapter 中的几个常用方法。

（1）函数原型：public void setDropDownViewResource (int resource)。

函数功能：创建一个下拉时的 view 的布局资源。

参数说明：resource，布局资源 id。

（2）函数原型：public T getItem (int position)。

函数功能：获取只能位置的元素。

参数说明：position 元素下标位置，从 0 开始。

返回值：获取到的元素。

（3）函数原型：public int getPosition (T item)。

函数功能：返回指定元素在数组中的位置。

参数说明：要获取位置的元素。

返回值：指定元素的位置。

（4）函数原型：public int getCount ()。

函数功能：返回数组中元素个数。

返回值：元素的个数。

（5）函数原型：public void add (T object)。

函数功能：添加指定的元素到数组的末尾。

参数说明：要添加的元素。

下面做一个动态添加、删除菜单项的例子。在窗体上添加一个 Spinner 组件用来显示城市信息，添加一个 EditText 用来显示当前选中项，以及输入要添加的新项。两个 Button 按钮，用来实现动态添加和删除项。

SpinnerExample2 .java 代码如下：

```
package com.SpinnerExample2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class SpinnerExample2 extends Activity {
    private Spinner mySpinner;
    private Button addBut;
    private Button removeBut;
    private EditText newCityEdit;
    private ArrayAdapter<String> adapter;
    private List<String> allCitys;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        addBut = (Button) this.findViewById(R.id.add);
                        //从 XML 布局文件中获取添加按钮对象
        removeBut = (Button) this.findViewById(R.id.remove);
                        //从 XML 布局文件中获取删除按钮对象
        newCityEdit = (EditText) this.findViewById(R.id.newCity);
                        //从 XML 布局文件中获取城市文本框对象
        allCitys = new ArrayList<String>();
                        //创建城市 ArrayList，并添加 3 个元素
        allCitys.add("北京");
        allCitys.add("上海");
        allCitys.add("深圳");
        mySpinner = (Spinner) this.findViewById(R.id.mySpinner);
                        //从 XML 布局文件中获取 Spinner 对象
        adapter = new ArrayAdapter<String>(this,
                  android.R.layout.simple spinner item, allCitys);
        //设置下拉菜单下拉项的布局
```

```
    adapter.setDropDownViewResource(android.R.layout.simple spinner dro-
    pdown item);mySpinner.setAdapter(adapter); //为 Spinner 添加适配器
    //添加按钮单击事件
    addBut.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            //TODO Auto-generated method stub
            String newCity = newCityEdit.getText().toString();
                                                //获取文本框中输入的城市
            for (int i = 0; i < adapter.getCount(); i++) {
                //判断当前选中的项和文本框中输入的是否相同
                if (newCity.equals(adapter.getItem(i))) {Toast.make-
                Text(SpinnerExample2.this, "该项已存在",Toast.LENGTH_
                SHORT).show();
                    return;
                }
            }
            if (!newCity.trim().equals("")) {  //文本框的内容不为“”时
                adapter.add(newCity);//将文本框中输入的信息添加到adapter中
                int position = adapter.getPosition(newCity);
                                   //获取 newCity 在 ArrayAdapter 中的位置
                mySpinner.setSelection(position);
                                   //选中下拉菜单中下标为 position 的项
                newCityEdit.setText("");        //清空文本框
            }
        }
    });
    removeBut.setOnClickListener(new OnClickListener() {
                                   //删除按钮单击事件
        @Override
        public void onClick(View v) {
            //TODO Auto-generated method stub
            if(mySpinner.getSelectedItem() != null) {
                   //从 adapter 中删除当前选中的项目
                adapter.remove(mySpinner.getSelectedItem().toString());
                newCityEdit.setText("");
                if (adapter.getCount() == 0) {
                                   //如果 adapter 中没有项目，提示用户
                    Toast.makeText(SpinnerExample2.this, "没有项目可以移除",
                            Toast.LENGTH SHORT).show();
                }
            }
        }
    });
    //当选择 mySpinner 中项目时触发该事件
    mySpinner.setOnItemSelectedListener(new OnItemSelectedListener() {
        @Override
        public void onItemSelected(AdapterView<?> parent, View view,
                int position, long id) {
            //TODO Auto-generated method stub
            //将当前选中的项目显示在 newCityEdit 上
            newCityEdit.setText(parent.getSelectedItem().toString());}
        @Override
        public void onNothingSelected(AdapterView<?> arg0) {
            //TODO Auto-generated method stub
         }
      });
    }
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFFFFFFF">
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="城市列表"
        android:textColor="#FF000000" />
    <Spinner android:id="@+id/mySpinner" android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="新增城市名称"
        android:textColor="#FF000000" />
    <EditText android:id="@+id/newCity" android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
    <LinearLayout android:orientation="horizontal"
        android:layout_width="fill_parent" android:layout_height="fill_parent">
        <Button android:id="@+id/add" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="添加"
            android:layout_gravity="center_horizontal" />
        <Button android:id="@+id/remove" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="删除"
            android:layout_gravity="center_horizontal" />
    </LinearLayout>
</LinearLayout>
```

运行效果如图 4.33 和图 4.34 所示。



图 4.33　为 Spinner 添加项



图 4.34　添加后的效果

4.16　相簿浏览——Gallery 的使用

Gallery 是图片浏览组件，主要实现横向显示图片列表。实现图片浏览效果，大致分为以下 4 步。

（1）初始化 Gallery 组件。

（2）创建一个新的 Adapter，继承 BaseAdapter，这个新的 Adapter 负责获取图片资源，例如图片名字，尺寸等信息。并通过重写 BaseAdapter 中的 getView 方法，实现设置图片显示的尺寸及显示方式。

（3）为 Gallery 组件添加 Adapter，该 Adapter 为新构建的 Adapter 类。

（4）通过 Gallery 的 setOnItemClickListener 方法实现单击 Gallery 中图片时的效果。

下面我们在 res/drawable 中添加四张图片，在窗体上添加一个 ImageView 组件和一个 Gallery 组件，当单击 Gallery 中图片时，在 ImageView 中显示图片。

GalleryExample .java 代码如下：

```
package com.GalleryExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class GalleryExample extends Activity {
     private Gallery myGallery;
     private ImageView myImg;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);
       myGallery=(Gallery) this.findViewById(R.id.mygallery);
                                       //从 XML 文件中获取 Gallery
       myImg=(ImageView) this.findViewById(R.id.myImg);
                                       //从 XML 文件中获取 ImageView
       try {
         myGallery.setAdapter(new ImageAdapter(this));
                                       //为 Gallery 添加适配器
       } catch (IllegalArgumentException e) {
           //TODO Auto-generated catch block
           e.printStackTrace();
       } catch (IllegalAccessException e) {
           // TODO Auto-generated catch block
           e.printStackTrace();
       }

       myGallery.setOnItemClickListener(new OnItemClickListener() {
                                       //Gallery 单击事件
          public void onItemClick(AdapterView parent, View v, int position,
          long id) {
            GalleryExample.this.setTitle(String.valueOf(position));
                                //将当前单击图片的位置显示在窗体标题栏
            try {
               //将当前单击的图片显示在 ImageView 中，imgList 是存储图片的集合
                  myImg.setImageResource(new ImageAdapter(GalleryExam
                  ple.this).myImgList.get(position).intValue());
               } catch (IllegalArgumentException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
               } catch (IllegalAccessException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
               }
          }
       });
    }
    private class ImageAdapter extends BaseAdapter{
       private Context mContext;
```

```
        private ArrayList<Integer> myImgList=new ArrayList<Integer>();
        private ArrayList<Object> myImgSize=new ArrayList<Object>();
        public ImageAdapter(Context c) throws IllegalArgumentException,
        IllegalAccessException{
            mContext = c;
            //获取资源中的图片 ID 和尺寸 ，通过反射机制来实现
            Field[] myFields = R.drawable.class.getDeclaredFields();
            for(int i=0;i<myFields.length;i+=1)
            {
                if (!"icon".equals(myFields[i].getName()))
                                                //除了 icon 之外的图片
                {
                    int index=myFields[i].getInt(R.drawable.class);
                                                //获取图片 ID
                    myImgList.add(index);       //保存图片 ID 到 myImgList 中
                    int size[]=new int[2];      //保存图片大小到 myImgSize 中
                    Bitmap bmImg=BitmapFactory.decodeResource(getReSources
                    (),index);
                    size[0]=bmImg.getWidth();
                    size[1]=bmImg.getHeight();
                    myImgSize.add(size);
                }
            }
        }
        @Override
        public int getCount() {                 //获取图片的个数
            // TODO Auto-generated method stub
              return myImgList.size();
        }
        @Override
        public Object getItem(int position) {
            // TODO Auto-generated method stub
            return position;
        }
        @Override
        public long getItemId(int position) {
            // TODO Auto-generated method stub
            return position;
        }
        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            // TODO Auto-generated method stub
            ImageView imageView = new ImageView (mContext);
            imageView.setImageResource(myImgList.get(position).intValue());
                                                //从 imgList 取得图片 ID
            imageView.setScaleType(ImageView.ScaleType.FIT_XY);
                                                //设置比例类型
            int size[]= new int[2];             //从 myImgSize 取得图片大小
            size=(int[]) myImgSize.get(position);
           //设置布局，图片原尺寸大小显示
         imageView.setLayoutParams(new Gallery.LayoutParams(size[0], size[1]));
            return imageView;
        }
    }
}
```

Rest/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFFFFFFF">
    <ImageView android:id="@+id/myImg" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:layout_weight="2"
        android:layout_gravity="center" />
    <!-- Gallery：图片浏览组件，android:spacing 设置图片的间距 -->
    <Gallery android:id="@+id/mygallery" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:spacing="10dp"
        android:layout_weight="1" />
</LinearLayout>
```

运行效果如图图 4.35 所示。

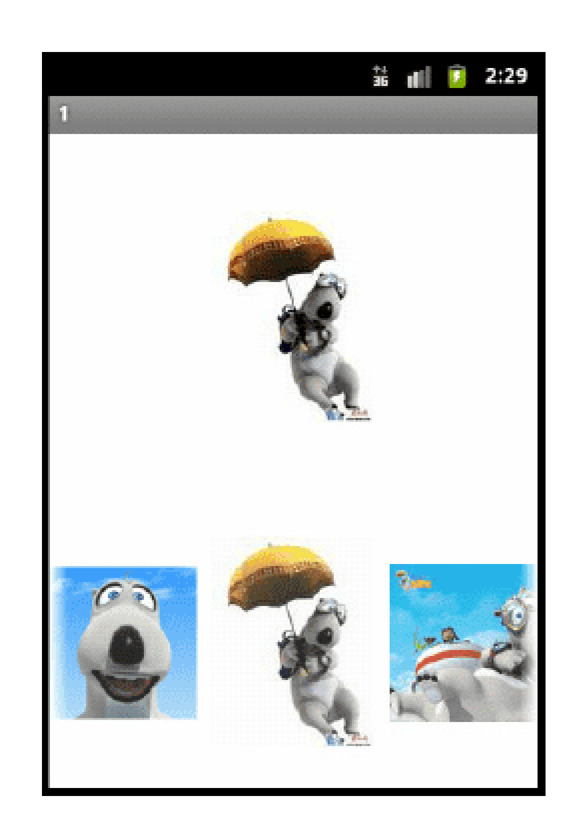


图 4.35　Gallery 示例

4.17　图片的缩放及旋转

在 Android 中提供了 Matrix 类，中文中叫矩阵，主要用于图片的缩放、平移、旋转等操作。可以通过 postRotate()方法实现图片旋转，postScale()方法实现图片的缩放，然后再重绘图片，即可达到缩放、旋转效果。下面我们看一个示例，在窗体上添加四个按钮，分别实现左旋转、右旋转、放大和缩小 4 个功能。

MatrixExample .java 代码如下：

```
package com.MatrixExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class MatrixExample extends Activity {
    private Bitmap myBitmap;                    //声明 Bitmap 类型变量
    private Matrix myMatrix = new Matrix();//声明并创建 Matrix 对象
    private int width;                          //声明 int 类型变量
    private int height;                         //声明 int 类型变量
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);      //加载布局文件
        Button rotateLeftBut = (Button) this.findViewById(R.id.rotateLeft);
                                            //获取 XML 文件中的左旋 Button
        Button rotateRightBut = (Button) this.findViewById(R.id.rotate-
        Right);                             //获取 XML 文件中的右旋 Button
        Button scaleBigBut = (Button) this.findViewById(R.id.scaleBig);
                                            //获取 XML 文件中的放大 Button
        Button scaleSmallBut = (Button) this.findViewById(R.id.scaleSmall);
                                            //获取 XML 文件中的缩小 Button
        //获取资源文件中的 p2 图片的 Bitmap,getResources()方法用来获取应用程序下的
        系统资源
        myBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.p2);
        //获取图片的原始的大小
        width = myBitmap.getWidth();
        height = myBitmap.getHeight();
        rotateLeftBut.setOnClickListener(new OnClickListener() {
                                                //左旋转 Button 的单击事件
```

```
                @Override
                public void onClick(View v) {
                    //TODO Auto-generated method stub
                    myMatrix.postRotate(-90);        //逆时针旋转 90 度
                    //创建一个新的图片，重新绘图
                    Bitmap newBitmap = Bitmap.createBitmap(myBitmap, 0, 0,
                            width, height, myMatrix, true);
                //建 Bitmap 转换为 Drawable 对象，使其可以使用在 ImageView 和
                ImageButton 中
                    BitmapDrawable newbmp = new BitmapDrawable(newBitmap);
                    //创建 ImageView 的对象
                    ImageView imageView = (ImageView) MatrixExample.this
                            .findViewById(R.id.pic);

                    imageView.setImageDrawable(newbmp);
                                                    //设置 ImageView 的背景图片
                }
            });
        rotateRightBut.setOnClickListener(new OnClickListener() {
                                                    //右旋转 Button 的单击事件
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                myMatrix.postRotate(90);        //顺时针旋转 90 度
                //创建一个新的图片，重新绘图
                Bitmap newBitmap = Bitmap.createBitmap(myBitmap, 0, 0, width,
                        height, myMatrix, true);
                //创建 Bitmap 转换为 Drawable 对象，使其可以使用在 ImageView 和
                ImageButton 中
                BitmapDrawable newbmp = new BitmapDrawable(newBitmap);
                //创建 ImageView 的对象
                ImageView imageView = (ImageView) MatrixExample.this
                        .findViewById(R.id.pic);
                imageView.setImageDrawable(newbmp);
                                                    //设置 ImageView 的背景图片
            }
        });
        scaleSmallBut.setOnClickListener(new OnClickListener() {
                                                    //缩 Button 的单击事件
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                //缩放图片的动作，宽高的缩放比例为 0.8
                myMatrix.postScale(0.8f, 0.8f);
                //创建一个新的图片，重新绘图
                Bitmap newBitmap = Bitmap.createBitmap(myBitmap, 0, 0, width,
                        height, myMatrix, true);
                //创建 Bitmap 转换为 Drawable 对象，使其可以使用在 ImageView 和
                ImageButton 中
                BitmapDrawable newbmp = new BitmapDrawable(newBitmap);
                //创建 ImageView 的对象
                ImageView imageView = (ImageView) MatrixExample.this
                        .findViewById(R.id.pic);
                imageView.setImageDrawable(newbmp);
                //设置 ImageView 的背景图片
            }
        });
        scaleBigBut.setOnClickListener(new OnClickListener() {
```

```
                                            //放大 Button 的单击事件
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                myMatrix.postScale(1.2f, 1.2f);
                                        //缩放图片的动作，宽高的缩放比例为 0.8
                //创建一个新的图片，重新绘图
                Bitmap newBitmap = Bitmap.createBitmap(myBitmap, 0, 0, width,
                        height, myMatrix, true);
                //创建 Bitmap 转换为 Drawable 对象，使其可以使用在 ImageView 和
                ImageButton 中
                BitmapDrawable newbmp = new BitmapDrawable(newBitmap);
                //创建 ImageView 的对象
                ImageView imageView = (ImageView) MatrixExample.this
                        .findViewById(R.id.pic);
                imageView.setImageDrawable(newbmp);
                                        //设置 ImageView 的背景图片
            }
        });
    }
}
```

Res/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout_height="fill_parent"  android:background="#FF929854">
    <LinearLayout android:orientation="horizontal"
        android:layout width="wrap content" android:layout height="wrap
        content"
        android:layout gravity="center">
        <Button android:id="@+id/rotateLeft" android:text="左旋"
            android:layout width="wrap content" android:layout height=
            "wrap content"
            android:layout_weight="1" />
        <Button android:id="@+id/rotateRight" android:text="右旋"
            android:layout width="wrap content" android:layout height=
            "wrap_content"
            android:layout weight="1" />
        <Button android:id="@+id/scaleBig" android:text="放大"
            android:layout width="wrap content" android:layout height=
            "wrap content"
            android:layout_weight="1" />
        <Button android:id="@+id/scaleSmall" android:text="缩小"
            android:layout width="wrap content" android:layout height=
            "wrap_content"
            android:layout weight="1" />
    </LinearLayout>
    <ImageView android:id="@+id/pic" android:layout width="wrap content"
        android:layout height="wrap content"
android:background="@drawable/p2"
        android:layout gravity="center" />
</LinearLayout>
```

运行效果如图 4.36 所示。

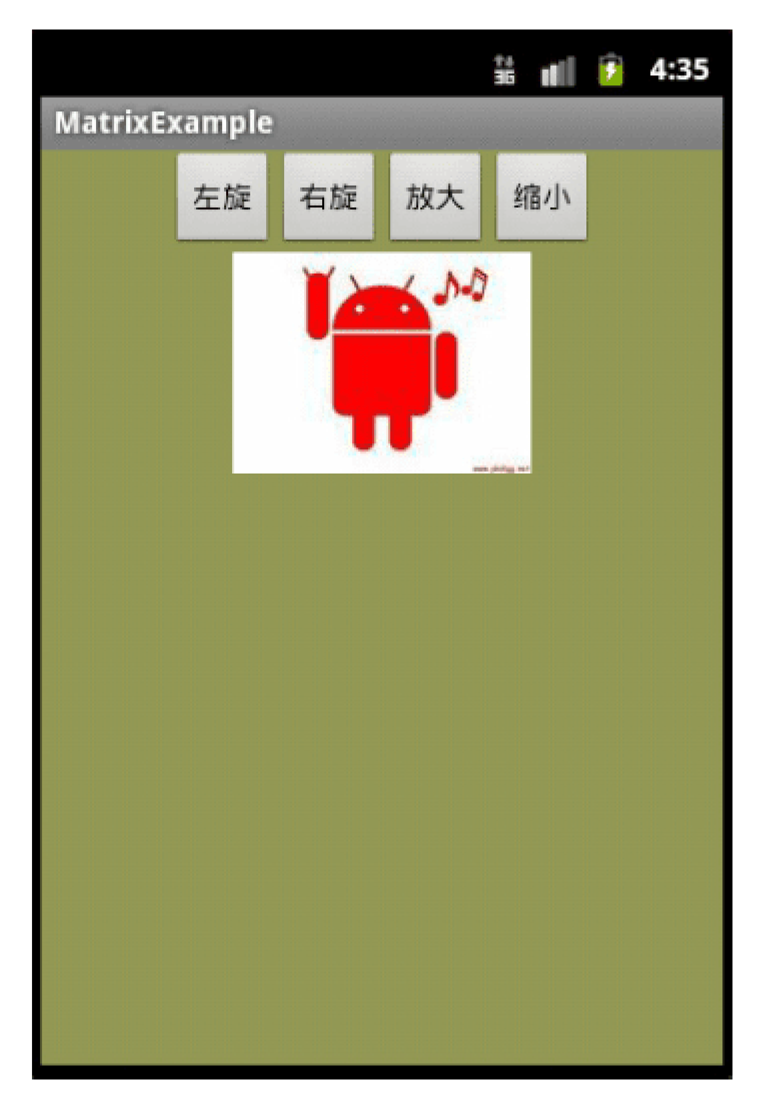


图 4.36　图片缩放和旋转

4.18　自动完成输入框自动提示功能的菜单——AutoCompleteTextView 的应用

当我们使用百度或者谷歌的时候，在输入框中输入一两个字后，就会自动出现提示信息，在 Android 中可以通过 AutoCompleteTextView 和 ArrayAdapter 配合使用，实现该效果。将要提示的信息预先保存到数组中，在将该数组放入到 ArrayAdapter 中，然后通过 AutoCompleteTextView.setAdapter 方法添加适配器，就大功告成了。我们一起来看下面的例子。

AutoCompleteTextViewExample .java 代码如下：

```
package com.AutoCompleteTextViewExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class AutoCompleteTextViewExample extends Activity {
    private AutoCompleteTextView myTextView;
                                //声明 AutoCompleteTextView 变量
    private String[] autoStr={"ab","abc","abcd","def"};
                                //声明 String 数组，存储提示信息
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);              //加载布局资源文件
        //获取布局资源文件中的 AutoCompleteTextView
        myTextView=(AutoCompleteTextView) this.findViewById(R.id.inputT-
        extView);
        //创建 ArrayAdapter 对象
        ArrayAdapter arrayAdapter=new ArrayAdapter(this,android.R.layout.-
        simple dropdown item 1line,autoStr);
        myTextView.setAdapter(arrayAdapter);    //为 myTextView 添加适配器
    }
```

```
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout width="fill parent"
    android:layout height="fill parent"
    >
    <!-- AutoCompleteTextView:具有提示信息的 TextView -->
    <AutoCompleteTextView android:id="@+id/inputTextView" android:layout
    width="fill_parent"
    android:layout_height="wrap_content" android:hint="请输入信息"
    ></AutoCompleteTextView>
</LinearLayout>
```

运行效果如图 4.37 所示。



图 4.37　AutoCompleteTextViewExample 示例

4.19　动态文字排版——GridView 网格视图实践

GridView 为网格视图，例如实现类似于九宫格效果，可以首选 GridView，该组件中的每一个条目通过 ListAdapter 和该组件进行关联。常用的 XML 属性如表 4.6 所示。

表 4.6　GridView的XML属性

属 性 名 称	描　　述
android：numColumns	列数。可以设置固定的列数，也可以设置为 auto_fit：自动填充。方法：setNumColumns(int)
android：columnWidth	设置列宽。方法：setColumnWidth(int)
android：stretchMode	缩放模式。方法：setStretchMode(int)
android：horizontalSpacing	两列之间的间距。方法：setHorizontalSpacing(int)

续表

属 性 名 称	描　　述
android：verticalSpacing	两行之间的间距。方法：setVerticalSpacing(int)
android：gravity	设置组件内容在组件中的位置。可选值为：top、bottom、left、right、center_vertical、fill_vertical、center_horizontal、fill_horizontal、center、fill、clip_vertical 可以多选，用“|”分开。关联方法：setGravity (int gravity)

下面通过例子了解 GridView 的使用，在本例中获得桌面应用程序图标，并将其显示在 GridView 中。

GridViewExample .java 代码如下：

```
package com.GridViewExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class GridViewExample extends Activity {
    private GridView myGridView;           //声明 GridView 类型变量
    private List<ResolveInfo> myAppIcon;   //声明变量，存放桌面应用程序图标
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);      //加载布局资源
        myGridView = (GridView) findViewById(R.id.myGrid);
                                           //获取资源文件中的 GridView 组件
        loadAppIcon();//加载桌面图标
        BaseAdapter adapter = new BaseAdapter() {
                                       //声明 BaseAdapter 对象，实现抽象方法
            @Override
            public int getCount() {     //项目个数
                //TODO Auto-generated method stub
                return myAppIcon.size();
            }
            @Override
            public Object getItem(int position) {  //获取指定位置的项目
                //TODO Auto-generated method stub
                return myAppIcon.get(position);
            }
            @Override
            public long getItemId(int position) {  //获取指定位置项目 id
                //TODO Auto-generated method stub
                return position;
            }
            //定义每一项显示的内容
            @Override
            public View getView(int position, View convertView, ViewGroup
            parent) {
                //TODO Auto-generated method stub
                ImageView imageView;
                if (convertView == null) {
                    imageView = new ImageView(GridViewExample.this);
                                                 //创建 ImageView 对象
                    //设置图片的填充方式，这里为按比例拉伸图片
                    imageView.setScaleType(ImageView.ScaleType.FIT CENTER);
                    //设置 imageView 的大小为 50*50
                    imageView
```

```
                        .setLayoutParams(new GridView.LayoutParams(50,50));
                } else {
                    imageView = (ImageView) convertView;
                }
                //获取 myAppIcon 中下标为 position 的 ResolveInfo
                ResolveInfo info = myAppIcon.get(position);
                //设置 imageView 显示的图片
                imageView.setImageDrawable(info.activityInfo
                        .loadIcon(getPackageManager()));
                return imageView;
            }
        };
        myGridView.setAdapter(adapter);         //为 myGridView 添加适配器
    }
    /**
     * 加载桌面图标
     */
    private void loadAppIcon() {
        Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
                                                //创建 Intent
        mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);
                                                //添加桌面应用程序列表到 Intent 中
        myAppIcon = getPackageManager().queryIntentActivities(mainIntent, 0);
    }
}
```

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myGrid" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:padding="10dp"
    android:verticalSpacing="10dp" android:horizontalSpacing="10dp"
    android:numColumns="auto_fit" android:columnWidth="60dp"
    android:stretchMode="columnWidth" android:gravity="center" />
```

运行效果如图 4.38 所示。



图 4.38　GridView 示例

4.20　列表的展示——ListView 的使用大全

ListView 是列表组件，是 Android 中很常用的组件。列表显示信息由以下 3 个部分组成：

- ListView 组件。
- 适配器，用来将数据映射到 ListView 组件中。
- 列表中要显示的数据。

ListView 适配器可以是 ArrayAdapter、SimpleAdapter 和 SimpleCursorAdapter。其中 ArrayAdapter 只显示一行文字；SimpleAdapter 可以自定义每行的数据显示形式；SimpleCursorAdapter 把数据库中的内容以列表的方式显示出来。

4.20.1　ListView 的使用——ArrayAdapter

本节通过 ArrayAdapter 构建一个简单的 ListView，每一个列表项显示一行文字。

ListViewExample1 .java 代码如下：

```
package com.ListViewExample1;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ListViewExample1 extends Activity {
    private LinearLayout myLayout                //声明 LinearLayout 类型变量
    private ListView myListView;                 //声明 ListView 类型变量
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);           //加载布局资源
        myLayout = (LinearLayout) this.findViewById(R.id.myLayout);
                                             //获取布局资源中的 LinearLayout
        myListView = new ListView(this);         //创建 ListView 对象
        //创建 ArrayAdapter 适配器。构造函数中的第一个参数含义是上下文 Context；第
        二个参数的含义是每一行的布局资源文件，android.R.layout.simple_exp ndable_
        list_item_1：系统定义好的，只显示一行文字；第三个参数的含义是数据源，是一个
        List 集合
        ArrayAdapter<String> adapter=new ArrayAdapter<String>(this,
                android.R.layout.simple_expandable_list_item_1, getMyData());
        myListView.setAdapter(adapter);      //为 myListView 添加适配器
        myLayout.addView(myListView);        //将myListView添加到myLayout上
    }
    /**
     * 获取数据
     * @return List
     */
    public List<String> getMyData() {
```

```
        //创建 List 对象，并向其添加数据
        List<String> myList = new ArrayList<String>();
        myList.add("数据项 1");
        myList.add("数据项 2");
        myList.add("数据项 3");
        myList.add("数据项 4");
        myList.add("数据项 5");
        return myList;
    }
}
```

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLayout" android:orientation="vertical"
    android:layout_width="fill_parent" android:layout_height="fill_parent">
</LinearLayout>
```

运行结果如图 4.39 所示。



图 4.39　ListView 示例 1

4.20.2　ListView 的使用——SimpleAdapter

SimpleAdapter 是简单且较为灵活的适配器，可以自定义 XML 显示每一个数据行。可以放图片、按钮、复选框、单选框等等。

本节示例展示了一个商品列表，本例和上例类似，自定义 XML 布局文件 listviewrow.xml 实现列表中每一行的布局，在构建 SimpleAdapter 对象时加载 listviewrow.xml 布局，并对相应的列表项信息赋值。我们一起来看看代码上的实现。

ListViewExample2 .java 代码如下：

```
package com.ListViewExample2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ListViewExample2 extends Activity {
```

```
private LinearLayout myListLayout;     //声明 LinearLayout 类型变量
private ListView tripListView;         //声明 ListView 类型变量
//创建 list 对象，用来存放列表项每一行的 Map 信息
List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
@Override
protected void onCreate(Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    this.setContentView(R.layout.main);        //加载布局资源
    myListLayout = (LinearLayout) this.findViewById(R.id.myListView);
                                               //获取 LinearLayout 对象
    tripListView = new ListView(this);         //创建 ListView 对象
    //创建布局参数
    LinearLayout.LayoutParams tripListViewParam = new LinearLayout.
    LayoutParams(
            LinearLayout.LayoutParams.FILL_PARENT,
            LinearLayout.LayoutParams.FILL_PARENT);
    //当拖曳列表时，显示的颜色，默认为黑色，这里设置为白色
    tripListView.setCacheColorHint(Color.WHITE);
     //将列表 tripListView 添加到流式布局 myListLayout 中
    myListLayout.addView(tripListView, tripListViewParam);
    //构建 SimpleAdapter 对象，构造函数共有 5 个参数，第一个参数的含义是上下文
    Context；第二个参数的含义是每一行的布局资源文件，这里自定义的列表项布局文
    件；第三个参数的含义是 HashMap 中的 key 信息 img，name，money，zhe；第四个
    参数的含义是 listviewrow.xml 中的组件 id；第五个参数的含义是 listviewrow.xml
    中的组件 id
    SimpleAdapter adapter = new SimpleAdapter(this, getTripListData(),
            R.layout.listviewrow, new String[] { "img", "name", "money",
                    "zhe" }, new int[] { R.id.tripImg, R.id.phoneName,
                    R.id.phoneMoney, R.id.phoneDiscount });
    tripListView.setAdapter(adapter);   //为列表 tripListView 添加适配器
    //列表项的单击事件
    tripListView.setOnItemClickListener(new OnItemClickListener() {
        /* 单击列表项时触发 onItemClick 方法，四个参数含义分别为
         * arg0：发生单击事件的 AdapterView
         * arg1：AdapterView 中被单击的 View
         * position：当前单击的行在 adapter 的下标
         * id：当前单击的行的 id
         */
        @Override
        public void onItemClick(AdapterView<?> arg0, View arg1,
                int position, long id) {
            // TODO Auto-generated method stub
            Toast.makeText(ListViewExample2.this,
                    "您选择的是" + list.get(position).get("name").toString(),
                    Toast.LENGTH_SHORT).show();
        }
    });
}

/**
 * 功能：获取列表项显示的数据
 * @return List
 */
public List<Map<String, Object>> getTripListData() {
    Map<String, Object> map = new HashMap<String, Object>();
                                             //创建 HashMap 对象
```

```
        map.put("img", R.drawable.moto);        //列表项图片
        map.put("name", "摩托罗拉（motorola）XT711 3G 手机"); //列表项手机名称
        map.put("money", "2699 元");              //列表项中手机价格
        map.put("zhe", "9 折");                   //列表项中手机折扣
        list.add(map);                            //将 map 添加到 list 中
        map = new HashMap<String, Object>();
        map.put("img", R.drawable.iphone);
        map.put("name", "iPhone4 16G 版");
        map.put("money", "5880 元");
        map.put("zhe", "8.5 折");
        list.add(map);
        map = new HashMap<String, Object>();
        map.put("img", R.drawable.samsung);
        map.put("name", "三星（SAMSUNG）i9003 3G 手机");
        map.put("money", "3099 元");
        map.put("zhe", "9 折");
        list.add(map);
        return list;
    }
}
```

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myListView" android:orientation="vertical" android:
    layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFFFF9EB">
</LinearLayout>
```

列表中每一行的布局文件 res/layout/listviewrow.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!--
android:background：设置当列表项获取焦点和按下时显示的效果，
trippoilistviewbg.xml 在 res/drawable/位置 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="@drawable/t-
    rippoilistviewbg">
    <!-- 用来显示手机图片 -->
    <ImageView android:id="@+id/tripImg" android:layout_width="68px"
        android:layout_height="65px" android:layout_margin="10px" />
    <!-- 手机名称，价格，打折信息的 LinearLayout-->
    <LinearLayout android:orientation="vertical"
        android:layout_width="wrap_content" android:layout_height="wrap_
        content"
        android:layout_marginTop="10px" android:layout_marginRight="10px"
        android:layout_marginBottom="10px">
        <!-- 手机名称 -->
        <TextView android:id="@+id/phoneName" android:layout_width=
        "wrap_content"
            android:layout_height="wrap_content" android:textColor="#ff-
            000000" />
        <!-- 手机价格 -->
        <TextView android:id="@+id/phoneMoney" android:layout_width="wrap_
        content"
            android:layout_height="wrap_content" android:textColor="#ff0
            00000"
```

```
            android:layout_marginTop="5px" android:layout_marginRight=
            "20px" />
        <!-- 手机打折 -->
        <TextView android:id="@+id/phoneDiscount" android:layout_width=
        "wrap_content"
            android:layout_height="wrap_content" android:textColor="#ff
            FF0000"
            android:layout_marginTop="5px" android:layout_marginRight=
            "20px" />
    </LinearLayout>
</LinearLayout>
```

列表项按下，获取焦点，选中时的效果在 res/drawable/trippoilistviewbg.xml 中，效果如图 4.41 所示。代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 设置按下时效果 -->
    <item android:state_pressed="true" android:drawable="@drawable/beijing" />
    <!-- 设置选中时效果 -->
    <item android:state_selected="true" android:drawable="@drawable/beijing" />
    <!-- 设置获取焦点时效果 -->
    <item android:state_focused="true" android:drawable="@drawable/beijing" />
</selector>
```

所有图片资源存放位置为 res/drawable/下。运行效果如图 4.40 和图 4.41 所示。




图 4.40　ListView 示例



图 4.41　选中列表中某一行效果

4.20.3　ListView 的使用——SimpleCursorAdapter

SimpleCursorAdapter 允许绑定一个游标的列到 ListView 上，并可以使用自定义的 layout 显示每个项目。SimpleCursorAdapter 的构造函数，如下：

```
public SimpleCursorAdapter (Context context, int layout, Cursor c, String[]
```

```
from, int[] to);
```

下面的例子是将通讯录中的联系人显示到 ListView 中。通讯录的信息通过 getContentResolver().query(People.CONTENT_URI, null, null, null, null)方法返回一个 Cursor 对象，然后将 Cursor 对象传递给 SimpleCursorAdapter 作为数据源。其中 query 方法的原型为：

```
public final Cursor query (Uri uri, String[] projection, String selection,
String[] selectionArgs, String sortOrder),
```

该方法是返回指定 Uri 的 Cursor，Cursor 是数据库每行的集合，关于数据库的介绍会在第 7 章详细介绍。该方法的参数含义如下。

- uri：由三部分组成："content：//"、数据的路径、标识 ID（可选项）。例如：content://contacts/people/ 用来返回设备上的所有联系人信息，content://contacts/people/8 表示联系人信息中 ID 为 8 的联系人记录。Android 中提供了一些辅助类，以类变量的形式给出查询字符串，例如 content://contacts/people/，通过 People.CONTENT_URI 获得。
- projection：返回指定列数据，如果是 null 表示返回全部列。
- selection：对行进行过滤的参数，类似于 SQL 中的 WHERE 语句（不包含 WHERE 关键字本身），如果是 null 表示返回所有行数据。
- selectionArgs：可选参数，在 selection 中可以包含"？"，这些值通过 selectionArgs 参数指定。
- sortOrder：如何排序行，类似于 SQL 中的 ORDER BY 语句（不包含 ORDER BY 关键字本身）。

例如查看联系人中名字是 Lisi 的人，可以通过下面的方式实现：

```
Cursor cursor = getContentResolver().query(People.CONTENT_URI, null,"NAME
= ?", new String[]{"Lisi"}, null)
```

下面我们看一下具体案例实现。

ListViewExample3 .java 代码如下：

```
package com.ListViewExample3;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ListViewExample3 extends Activity {
     private ListView listView;             //声明 ListView 变量
     private LinearLayout myLayout;         //声明 LinearLayout 变量

     /** Called when the activity is first created. */
     @Override
     public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.main);   //加载布局资源
          myLayout = (LinearLayout) this.findViewById(R.id.myLayout);
                                           //获取 LinearLayout 对象
          listView = new ListView(this);   //创建 ListView 对象
          //获取通讯录数据库中的数据的 Cursor 对象，具体数据库操作将在第 7 章详细介绍
         Cursor cursor = getContentResolver().query(People.CONTENT_URI, null,
                   null, null, null);
         //将 Cursor 交给 Activity 管理，这样 Cursor 生命周期可以和 Activity 自动同步
          startManagingCursor(cursor);
          //创建 SimpleCursorAdapter 对象。构造函数有 5 个参数，第一个参数的含义是
```

```
            上下文 Context；第二个参数的含义是每一行的布局资源文件，android.R.layout.-
            simple_expandable_list_item_1 是系统定义好的，只显示一行文字；第三个
            参数的含义是 Cursor 对象作为数据源；第四个参数的含义是 String 数组，数据库
            的字段信息；第五个参数的含义是 int 数组，包含布局文件对应的 id
            ListAdapter listAdapter = new SimpleCursorAdapter(this,
                   android.R.layout.simple_expandable_list_item_1, cursor,
                    new String[] { People.NAME }, new int[] { android.
                    R.id.text1 });
        listView.setAdapter(listAdapter);      //为列表 listView 添加适配器
        myLayout.addView(listView);   //将列表 listView 添加到布局 myLayout 上
    }
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLayout" android:orientation="vertical"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:background="#FF666666">
</LinearLayout>
```

运行效果如图 4.42 所示。



图 4.42　ListView 示例

4.21　选项菜单——OptionsMenu

菜单在 Android 中使用比较频繁，Android 中菜单分为 3 种，即选项菜单（OptionsMenu）、上下文菜单（ContextMenu）和子菜单（SubMenu）。这一节我们研究一下选项菜单（OptionsMenu）。当按下手机上的 Menu 键时，每个 Activity 都可以重写 onCreateOptionsMenu(Menu menu)方法，实现在屏幕底端弹出菜单，这个菜单被称为选项菜单（OptionsMenu）。一般选项菜单（OptionsMenu）最多可以有两行，每行可以有 3 个菜单项，当多于 6 项时，第 6 项位置会出现一个 more。单击 more 可以看到其他的菜单项。

选项菜单（OptionsMenu）上面可以有文字和图片，另外当单击某一个菜单项时，会触发 Activity 中的 onOptionsItemSelected(MenuItem item)方法，通过 item.getItemId()获取当前单击的菜单项的 id，实现单击不同菜单时的后续处理工作。

下面的例子在 Activity 中添加了有 7 个菜单项的菜单。并对菜单的相关事件进行捕获。

OptionsMenuExample .java 代码如下：

```
package com.OptionsMenuExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class OptionsMenuExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);                //加载资源文件
    }
    //单击 Menu 时，系统会调用该方法，初始化选项菜单（OptionsMenu）
    //并传入一个 menu 对象供你使用
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        //add()方法是添加一个新的菜单项到 menu 中，有 4 个参数，第一个参数的含义是组
        标识，如果不分组的话值为 Menu.NONE；第二个参数的含义是菜单项 id，是菜单项的唯
        一标识，可以通过它判断操作了哪个菜单项；第三个参数的含义是菜单项摆放的顺序；第四个
        参数的含义是菜单项上显示的文字
        MenuItem homeMenuItem = menu.add(Menu.NONE, 0, 0, "主页");
                                                      //添加菜单项
        homeMenuItem.setIcon(R.drawable.home);        //为菜单项设置图标
        MenuItem printMenuItem = menu.add(Menu.NONE, 1, 1, "打印");
        printMenuItem.setIcon(R.drawable.print);
        MenuItem saveMenuItem = menu.add(Menu.NONE, 2, 2, "保存");
        saveMenuItem.setIcon(R.drawable.save);
        MenuItem searchMenuItem = menu.add(Menu.NONE, 3, 3, "搜索");
        searchMenuItem.setIcon(R.drawable.search);
        MenuItem delMenuItem = menu.add(Menu.NONE, 4, 4, "删除");
        delMenuItem.setIcon(R.drawable.del);
        MenuItem settingMenuItem = menu.add(Menu.NONE, 5, 5, "设置");
        settingMenuItem.setIcon(R.drawable.setting);
        MenuItem aboutMenuItem = menu.add(Menu.NONE, 6, 6, "关于");
        aboutMenuItem.setIcon(R.drawable.about);
        return super.onCreateOptionsMenu(menu);
    }

    //菜单项被选择触发该方法
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        super.onOptionsItemSelected(item);

        switch (item.getItemId()) {                   //获取菜单项的 id
        case 0:
            //item.getTitle()：获取菜单项上显示的文字
            Toast.makeText(this,"单击了’"+item.getTitle()+"‘菜单",Toast.
            LENGTH_LONG).show();
            break;
        case 1:
            Toast.makeText(this,"单击了‘"+item.getTitle()+"’菜单",Toast.
            LENGTH_LONG).show();
```

```
                break;
            case 2:
                Toast.makeText(this, "单击了‘"+item.getTitle()+"’菜单", Toast.
                LENGTH_LONG).show();
                break;
            case 3:
                Toast.makeText(this, "单击了‘"+item.getTitle()+"’菜单", Toast.
                LENGTH_LONG).show();
                break;
            case 4:
                Toast.makeText(this, "单击了‘"+item.getTitle()+"’菜单", Toast.
                LENGTH_LONG).show();
                break;
            case 5:
                Toast.makeText(this, "单击了‘"+item.getTitle()+"’菜单", Toast.
                LENGTH_LONG).show();
                break;
            case 6:
                Toast.makeText(this, "单击了‘"+item.getTitle()+"’菜单", Toast.
                LENGTH_LONG).show();
                break;
            }
            return true;
        }
        //选项菜单（OptionsMenu）被关闭时触发该方法，3 种情况下选项菜单（OptionsMenu）
        会被关闭，即 back 按钮被单击、menu 按钮被再次按下、选择了某一个菜单项
        @Override
        public void onOptionsMenuClosed(Menu menu) {
            Toast.makeText(this, "选项菜单（OptionsMenu）被关闭了", Toast.
            LENGTH_LONG).show();
        }
        //选项菜单（OptionsMenu）显示之前调用该方法
        //返回值：false：此方法就把用户单击 menu 的动作给屏蔽了，onCreateOptionsMenu
        方法将不会被调用
        @Override
        public boolean onPrepareOptionsMenu(Menu menu) {
            Toast.makeText(this,
                    "选项菜单（OptionsMenu）显示之前 onPrepareOptionsMenu 方法会被调用",
                    Toast.LENGTH_LONG).show();
            return true;
        }
}
```

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFF0F0F0">
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="选项菜单（Optio-
        nsMenu）示例"
        android:textColor="#FF000000"/>
</LinearLayout>
```

菜单项的图片资源在 res/drawable 中。启动 Activity 后，单击 menu 按键，会调用 onPrepareOptionsMenu()方法，运行效果如图 4.43 所示。onPrepareOptionsMenu()方法调用

后，Activity 会调用 onCreateOptionsMenu()方法初始化菜单，多于 6 个菜单项时，会出现 More 菜单。如图 4.44 所示。



图 4.43　调用 onPrepareOptionsMenu 方法



图 4.44　选项菜单（OptionsMenu）效果

4.22　上下文菜单——ContextMenu

上下文菜单 ContextMenu 继承了 Menu 接口，选项菜单是服务于 Activity，上下文菜单是注册到某一个 View 对象上，如果 View 对象注册了上下文菜单，在该 View 上长按（约 2 秒），会显示上下文菜单。可以为上下文菜单指定标题文字及标题图标，但菜单选项不能附带图标，也不支持快捷键。上下文菜单的一个典型例子是长按通讯录列表某个联系人，会出现上下文菜单，显示操作信息。

创建一个上下文菜单，必须重写 Activity 的 onCreateContextMenu()方法初始化上下文菜单，及 onContextItemSelected()方法监听当选择上下文菜单的某一菜单项时，做不同的处理。然后通过 registerForContextMenu()方法为这个 View 注册一个上下文菜单。

下面的例子，当长按窗体时，出现上下文菜单，用来改变窗体的背景颜色。

ContextMenuExample .java 代码如下：

```
package com.ContextMenuExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ContextMenuExample extends Activity {
    private LinearLayout myLayout;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
         setContentView(R.layout.main);
         myLayout = (LinearLayout) this.findViewById(R.id.myLayout);
```

```
                                    //获取 LinearLayout 对象
        this.registerForContextMenu(myLayout);
                                    //为 myLayout 注册 ContextMenu 事件
    }
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,
            ContextMenuInfo menuInfo) {

        //TODO Auto-generated method stub
        super.onCreateContextMenu(menu, v, menuInfo);
        menu.setHeaderIcon(R.drawable.tinfo);  //设置上下文菜单标题的图标
        menu.setHeaderTitle("设置背景颜色");      //设置上文菜单的标题
        MenuItem homeMenuItem = menu.add(Menu.NONE, 0, 0, "绿色");
                                            //添加菜单项
        MenuItem printMenuItem = menu.add(Menu.NONE, 1, 1, "蓝色");
    }

    @Override
    public boolean onContextItemSelected(MenuItem item) {
        // TODO Auto-generated method stub
        switch (item.getItemId()) {            //获取菜单项的 id
        case 0:
            // item.getTitle():获取菜单项上显示的文字
            Toast.makeText(this, "单击了' " + item.getTitle() + " '菜单",
                    Toast.LENGTH_LONG).show();
            myLayout.setBackgroundColor(Color.GREEN);      // 设置背景颜色
            break;
        case 1:
            Toast.makeText(this, "单击了 '" + item.getTitle() + "' 菜单",
                    Toast.LENGTH_LONG).show();
            myLayout.setBackgroundColor(Color.BLUE);
            break;
        }
        return super.onContextItemSelected(item);
    }

}
```

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLayout" android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#FFFFFFFF">
</LinearLayout>
```

运行效果如图 4.45 所示。



图 4.45　ContextMenu 示例

4.23　子菜单——SubMenu

可以在 Menu 上添加子菜单（SubMenu），但子菜单不能再嵌套子菜单，即意味着 Android 中菜单只有两层，这是项目设计时需要注意的。可以在选项菜单或者上下文菜单中添加子菜单，子菜单项不支持显示图标。Menu 中提供了方法 addSubMenu()用来添加具有子菜单的菜单项。

下面的示例中为 OptionMenu 添加了子菜单“File”和“Edit”，选择“File”或“Edit”出现二级菜单。

SubMenuExample .java 代码如下：

```
package com.SubMenuExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class SubMenuExample extends Activity {
      /** Called when the activity is first created. */
      @Override
      public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           setContentView(R.layout.main);//加载 main.xml 资源文件
      }
      //单击 Menu 时，系统会调用该方法，初始化选项菜单（OptionsMenu）
      //并传入一个 menu 对象供你使用
      @Override
      public boolean onCreateOptionsMenu(Menu menu) {
             //TODO Auto-generated method stub
             SubMenu fileMenu = menu.addSubMenu(1,1,1,"File");
                                             //给 menu 添加子菜单
             fileMenu.setHeaderIcon(R.drawable.file);
                                             //设置子菜单弹出框的标题图标
```

```
        fileMenu.setHeaderTitle("File");   //设置子菜单弹出框的标题文字
        fileMenu.setIcon(R.drawable.file); //设置子菜单的图标

        fileMenu.add(2,11,11,"New");       //为子菜单添加二级菜单
        fileMenu.add(2,12,12,"Save");      //为子菜单添加二级菜单
        fileMenu.add(2,13,13,"Close");     //为子菜单添加二级菜单
        SubMenu editMenu = menu.addSubMenu(1,2,2,"Edit");
        EditMenu.setHeaderIcon(R.drawable.edit);
        editMenu.setHeaderTitle("Edit");
        editMenu.setIcon(R.drawable.edit);
        editMenu.add(2,21,21,"Redo");
        editMenu.add(2,22,22,"Undo Typing");
        return super.onCreateOptionsMenu(menu);
    }
    //菜单项被选择触发该方法
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        //TODO Auto-generated method stub
        switch(item.getItemId()){
        case 1:
            Toast.makeText(this, "单击了’"+item.getTitle()+"‘菜单", Toast.
            LENGTH LONG).show();
            break;
        case 2:
            Toast.makeText(this, "单击了’"+item.getTitle()+"‘菜单", Toast.
            LENGTH LONG).show();
            break;
        case 11:
            Toast.makeText(this, "单击了 File 子菜单’"+item.getTitle()+"‘",
            Toast.LENGTH LONG).show();
            break;
        case 12:
            Toast.makeText(this, "单击了 File 子菜单’"+item.getTitle()+"‘",
            Toast.LENGTH LONG).show();
            break;
        case 13:
            Toast.makeText(this, "单击了 File 子菜单’"+item.getTitle()+"‘",
            Toast.LENGTH LONG).show();
            break;
        case 21:
            Toast.makeText(this, "单击了 Edit 子菜单’"+item.getTitle()+"‘",
            Toast.LENGTH_LONG).show();
            break;
        case 22:
            Toast.makeText(this, "单击了 Edit 子菜单’"+item.getTitle()+"‘",
            Toast.LENGTH LONG).show();
            break;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout_height="fill_parent" android:background="#FFF0F0F0">
    <TextView android:layout_width="fill_parent"
```

```
        android:layout height="wrap content" android:text="SubMenu 示例" />
</LinearLayout>
```

运行效果如图 4.46 所示。单击“File”菜单，出现 File 的子菜单，如图 4.47 所示。

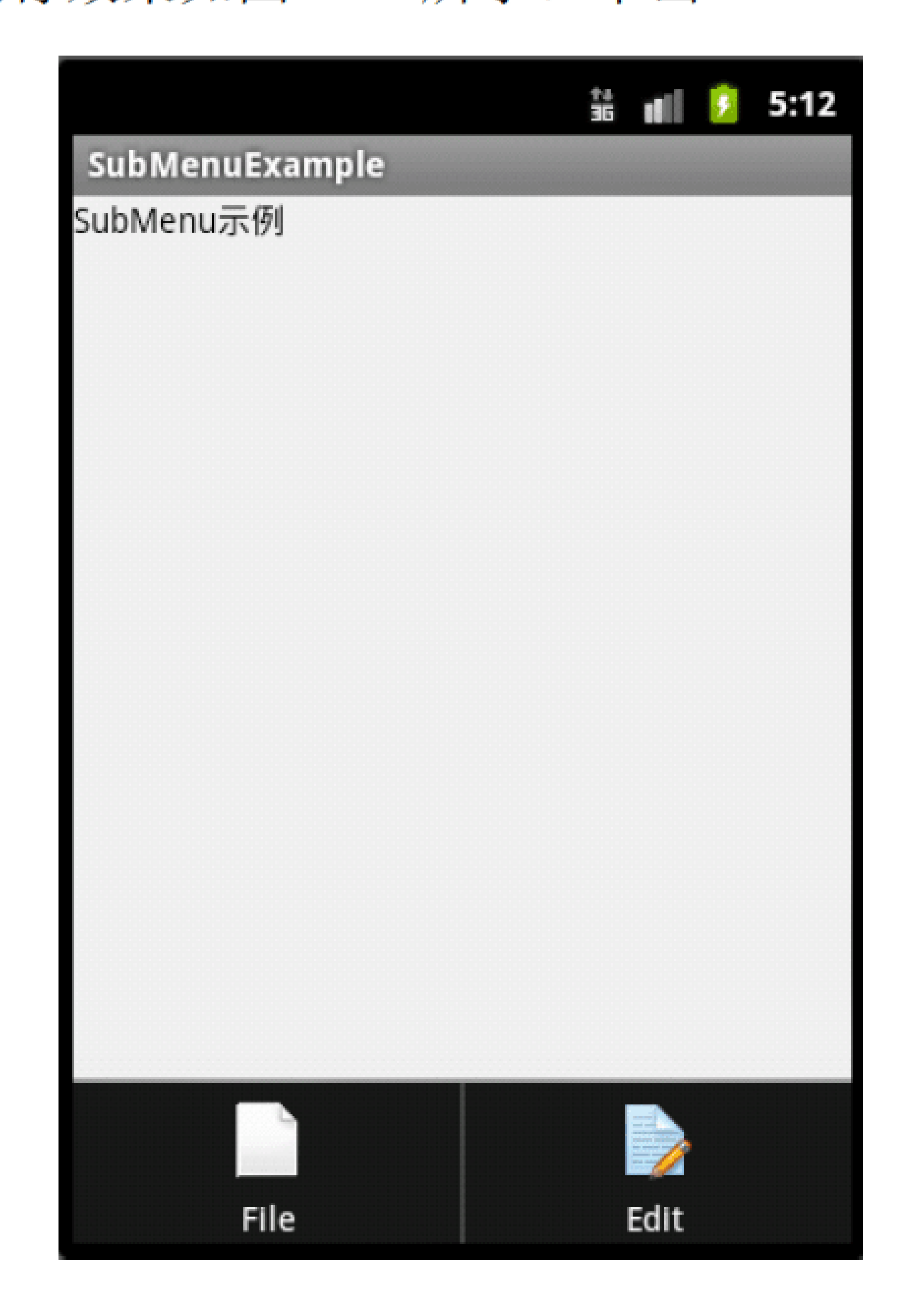



图 4.46　子菜单示例图 1



图 4.47　子菜单示例图 2

4.24　与用户交互的对话框——AlertDialog

Dialog 是 Android 对话框的基类，AlertDialog Dialog 的直接子类，可以通过 AlertDialog.Builder 在创建时指定对话框的内容和对话框的样式，Dialog 也可以使用自定义的 layout 作为对话框的显示内容。下面通过例子了解一下 AlertDialog 的实现。

AlertDialogExample .java 代码如下：

```
package com.AlertDialogExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class AlertDialogExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);     //加载资源文件
        final Button btnQuit = (Button) findViewById(R.id.okCancelBut);
                                           //获取资源文件中的 Button
        btnQuit.setOnClickListener(new Button.OnClickListener() {
                                           //btnQuit 单击事件
            public void onClick(View v) {
                new AlertDialog.Builder(AlertDialogExample.this)
                                           //创建 AlertDialog.Builder 对象
                        .setTitle("标题")   //设置标题
                        .setMessage("确定要退出吗？")      //设置显示信息
```

```
                    .setIcon(R.drawable.icon)  //设置标题栏显示的图标
                    //添加确定按钮及事件
                    .setPositiveButton("确定",new DialogInterface.On-
                    ClickListener(){
                            public void onClick(DialogInterface
                            dialog,int whichButton) {
                                setResult(RESULT OK);
                                finish();
                            }
                        })
                    //添加取消按钮及事件
                    .setNegativeButton("取消",new DialogInterface.
                    OnClickListener() {
                            public void onClick(DialogInterface dialog,
                                    int whichButton) {
                            }
                        }).show();      //显示对话框
        }
    });
    final  Button  btnTravels  =  (Button)  findViewById(R.id.listBut);
                                        //获取资源文件中的 Button
    btnTravels.setOnClickListener(new Button.OnClickListener() {
                                        //btnTravels 单击事件
        public void onClick(View v) {
            new AlertDialog.Builder(AlertDialogExample.this)
                                    //创建 AlertDialog.Builder 对象
                    .setTitle("导入导出联系人")       //设置标题
                    //设置对话框中显示的 list，这里显示内容存放到 arrcontent
                    数组中
                    .setItems(R.array.arrcontent,new DialogInterface.
                    OnClickListener() {
                            public void onClick(DialogInterface
                            dialog,int whichcountry) {//whichcoun-
                            try:当前单击的 Button 的 id 或者列表的下标
                                String[] travelcountries = getRes-
                                ources().getStringArray(R.array.
                                arrcontent);
                                //将资源文件中的 arrcontent 转换为数组
                                //单击列表对话框中的每一项弹出的对话框
                                new AlertDialog.Builder(AlertDia
                                logExample.this)
                                        .setMessage("你单击了: " +
                                            travelcountries[which
                                            country])//设置显示信息
                                        .setNeutralButton("取消",
                                        new DialogInterface.OnCli-
                                        ckListener() {//设置单击取消
                                        按钮事件
                                                public void on-
                                                Click(Dialog-
                                                Interface dia-
                                                log,int whichB
                                                utton) {
                                                }
                                            }).show();//显示对
                                            话框
                            }})
                    .setNegativeButton("取消", new DialogInterface.On-
```

```
                        ClickListener() {
                                    @Override
                                    public void onClick(DialogInterface
                                    dialog, int which) {
                                        // TODO Auto-generated method stub

                                    }
                                }).show();
                }
            });

        Button button=(Button)findViewById(R.id.defineBut);
                                            //获取资源文件中的 Button
        button.setOnClickListener(new Button.OnClickListener() {
                @Override
                public void onClick(View v) {
                    // TODO Auto-generated method stub
                    //创建 Dialog 对象,构造函数中的参数一是 Context，参数二是对话
                    框的样式，定义在 style.xml 中
                    final Dialog dialog = new Dialog(AlertDialogExample.
                    this,R.style.dialog);
                    dialog.setContentView(R.layout.myalertdialog);
                                            //设置对话框的布局资源文件
                                            //获取对话框上的按钮
                    final ImageButton cancelBut=(ImageButton) dialog.find-
                    ViewById(R.id.cancel);
                    cancelBut.setOnClickListener(new OnClickListener(){
                                                //"取消"按钮的单击事件
                        @Override
                        public void onClick(View v) {
                            // TODO Auto-generated method stub

                            dialog.hide();          //隐藏对话框
                        }

                    });
                    dialog.show();

                }

            });
    }
}
```

res/layout/main.xml 代码如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res/com.android.gif"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout_height="fill_parent" android:background="#FFFFFFFF">
    <Button android:id="@+id/okCancelBut" android:layout width="wrap
    content"
        android:layout_height="wrap_content" android:text="带 ok,cancel 按
        钮的 AlertDialog" />
    <Button android:id="@+id/listBut" android:layout width="wrap content"
        android:layout_height="wrap_content" android:text="带列表的 Alert
        Dialog" />
    <Button android:id="@+id/defineBut" android:layout width="wrap Content"
        android:layout_height="wrap_content" android:text="自定义的 AlertDialog" />
```

```
</LinearLayout>
```

自定义 Dialog 的资源文件 res/layout/myalertdialog.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="280px"
    android:layout_height="200px" android:id="@+id/myalert"
    android:padding="20px">
    <LinearLayout android:orientation="horizontal"
        android:layout_width="wrap_content" android:layout_height="wrap_
        content">
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="激活："
            android:textColor="#FFBFBFBF" android:gravity="center" />
        <TextView android:id="@+id/tripTitle" android:layout_width="wrap_
        content"
            android:layout_height="wrap_content" android:text="手机充值卡业务"
            android:textColor="#FFFFFFFF" android:gravity="center"
            android:textSize="14px" />
    </LinearLayout>
    <EditText android:id="@+id/inputKey" android:layout_width="240px"
        android:layout_height="40sp" android:hint="输入充值卡密码" android:
        textSize="15px"
        android:layout_marginTop="10px" android:numeric="integer" />
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="您可以淘宝商店购买："
        android:layout_marginTop="10px" android:textColor="#FFFFFFFF" />
    <LinearLayout android:orientation="horizontal"
        android:layout_width="fill_parent" android:layout_height="wrap_
        content"
        android:layout_marginTop="10px">
        <ImageButton android:id="@+id/cancel"
            android:layout_width="wrap_content" android:layout_height=
            "wrap_content"
            android:background="@drawable/qx" />
        <ImageButton android:id="@+id/ok" android:layout_width="wrap_
        content"
            android:layout_height="wrap_content" android:background="@d-
            rawable/qr"
            android:layout_marginLeft="8px" />
        <ImageButton android:id="@+id/buy" android:layout_width="wrap_
        content"
            android:layout_height="wrap_content" android:background="@d-
            rawable/gm"
            android:layout_marginLeft="8px" />
    </LinearLayout>
</LinearLayout>
```

res/values/arrays.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- 自定义数组，通过 name 调用该数组 -->
   <string-array name="arrcontent">
      <item>从 SIM 卡导入</item>
      <item>从 SD 卡导入</item>
      <item>导出到 SD 卡</item>
   </string-array>
</resources>
```

res/values/style.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="dialog" parent="@android:style/Theme.Dialog">
         <item name="android:windowFrame">@null</item><!--Dialog 的边框，
         @null 标识无-->
         <item name="android:windowIsFloating">true</item><!--是否浮现在
         activity 之上-->
         <item name="android:windowIsTranslucent">false</item><!--半透明-->
         <item name="android:windowNoTitle">true</item><!-- 无标题 -->
         <item name="android:windowBackground">@null</item><!-- 无背景颜色 -->
         <item name="android:backgroundDimEnabled">true</item><!--模糊-->
     </style>
</resources>
```

当单击“带 ok，cancel 按钮的 AlertDialog”按钮，运行效果如图 4.48 所示。当单击“带列表的 AlertDialog”按钮，运行效果如图 4.49 所示。当单击“自定义的 AlertDialog”按钮，运行效果如图 4.50 所示。



图 4.48　简单的 AlertDialog



图 4.49　带列表的 AlertDialog

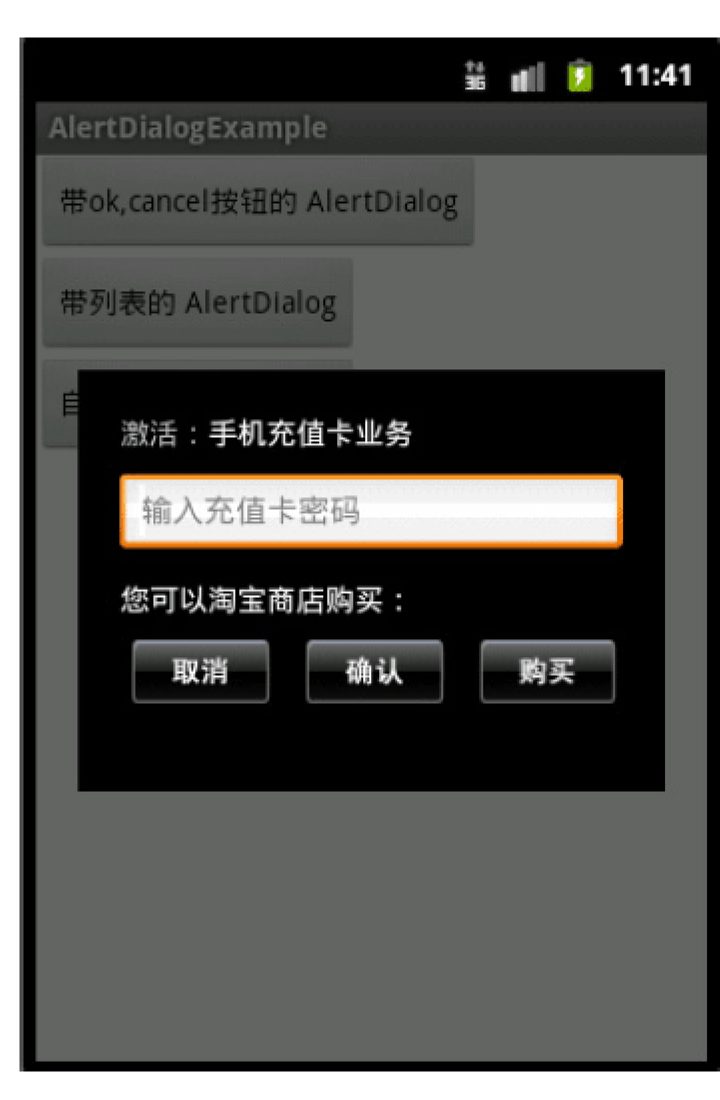


图 4.50　自定义的 Dialog

4.25　拖动条——SeekBar

在前面我们介绍过 ProgressBar 组件，拖动条外观和 ProgressBar 组件类似，一般我们在听歌或者看电影的时候，我们经常习惯快进一段或者回退一段，这时候就可以了解一下 SeekBar 的使用了。SeekBar 通过 SeekBar.OnSeekBarChangeListener 接口监听拖动条被拖动的事件。该接口中有 3 个抽象方法，分别为拖动条数值改变方法 onProgressChanged、开始拖动方法 onStartTrackingTouch 和停止拖动方法 onStopTrackingTouch。我们一起通过下面的示例了解 SeekBar 的使用。

SeekBarExample .java 代码如下：

```
package com.SeekBarExample;
```

```
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class SeekBarExample extends Activity {
    private SeekBar mySeekBar;          //声明 SeekBar 类型变量
    private TextView myProgressText;    //声明 TextView 类型变量
    private TextView myTrackingText;    //声明 TextView 类型变量
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); //加载资源文件
        myProgressText = (TextView) findViewById(R.id.myProgress);
                                        //获取资源文件中的 TextView
        myTrackingText = (TextView) findViewById(R.id.myTracking);
                                        //获取资源文件中的 TextView
        mySeekBar = (SeekBar) findViewById(R.id.mySeek);
                                        //获取资源文件中的 SeekBar
        mySeekBar.setOnSeekBarChangeListener(new  OnSeekBarChangeListener()
        {//拖动条改变事件
                    //拖动条改变时触发该事件，参数一为当前操作的拖动条，参数二为当
                    前拖动条的值
                    @Override
                    public void onProgressChanged(SeekBar seekBar,
                            int progress, boolean fromUser) {
                        //TODO Auto-generated method stub
                        myProgressText.setText(progress + " "
                                + "onProgressChanged" + "=" + fromUser);
                    }
                    //开始拖动拖动条时触发该方法
                    @Override
                    public void onStartTrackingTouch(SeekBar seekBar) {
                        //TODO Auto-generated method stub
                        myTrackingText.setText("TrackingTouch  Start");
                    }
                    //停止拖动拖动条时触发该方法
                    @Override
                    public void onStopTrackingTouch(SeekBar seekBar) {
                        //TODO Auto-generated method stub
                        myTrackingText.setText("TrackingTouch  Stop");
                    }
                });
    }
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <SeekBar android:id="@+id/mySeek" android:layout_width="fill_parent"
        android:layout height="wrap content" android:max="100"
        android:progress="30" android:secondaryProgress="50" />
    <TextView android:id="@+id/myProgress" android:layout width="match
    parent"
        android:layout height="wrap content" />
    <TextView android:id="@+id/myTracking" android:layout width="match
    parent"
        android:layout height="wrap content" />
</LinearLayout>
```

运行结果如图 4.51 所示。

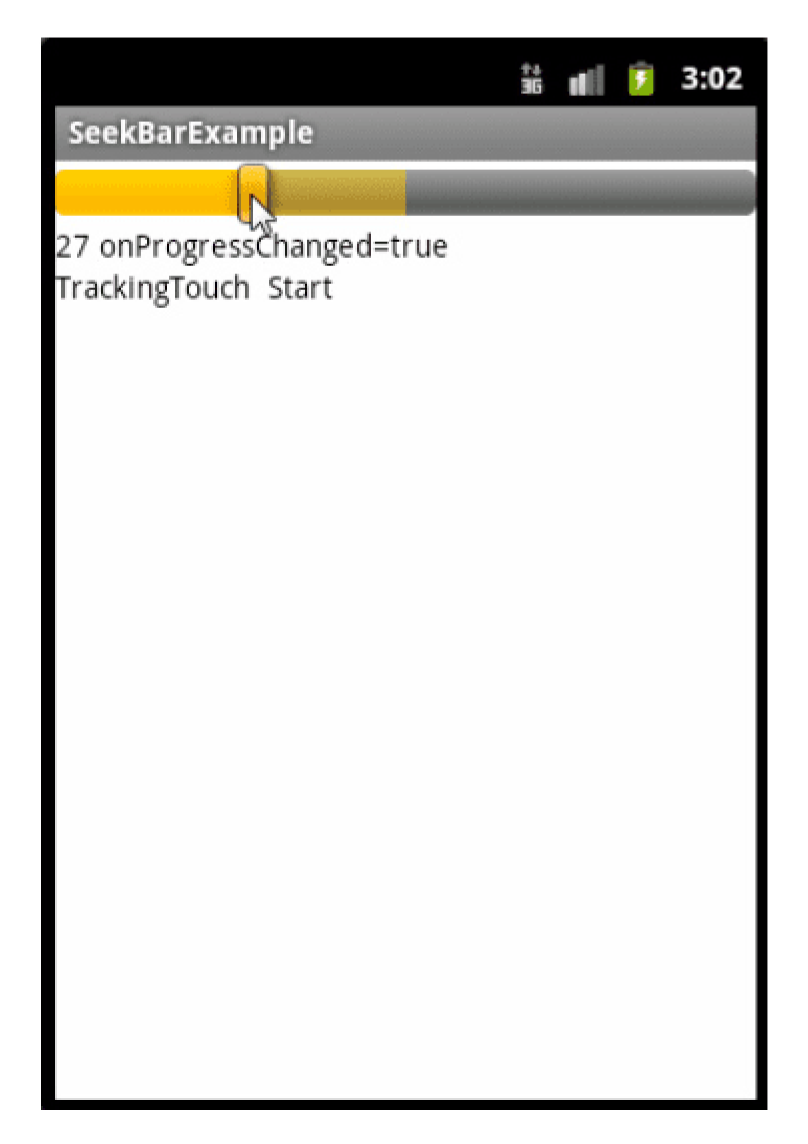


图 4.51　SeekBar 示例

4.26　使用主题——Theme

前面我们讲过 style，style 是一个包含一种或者多种组件属性的集合，XML 元素可以应用该 style，例如为 View 元素定义特定的颜色、字号。Theme 也是一种包含一种或者多种组件属性的集合，Theme 可以应用在某一个 Activity 或者全部 Activity 中，例如我们可以定义一个 Theme，该 Theme 可以对每个 Activity 的前景、背景、字体、字号等设置。定义 Theme 的方式和 style 类似，通过<style>标签定义，<style>标签内部，声明一个或者多个<item>，每个<item>可以定义一个属性名字及其对应值，为<style>标签添加一个全局唯一的名字，可以在 AndroidManifest.xml 中通过<application android:theme="@style/自定义 Theme 名字">为所有 Activity 添加该 Theme。也可以在 AndroidManifest.xml 中通过<activity android:theme="@ style/自定义 Theme 名字">为某一个 Activity 添加该 Theme，另外也可以在代码中通过 setTheme(R.style.CustomTheme)加载指定主题，R.style.CustomTheme 为 Theme 资源名称。

下面的例子，自定义了一个 Theme，Theme 中定义了文字颜色、背景颜色和文字大小。Activity 通过 setTheme()方法加载该 Theme。

ThemeExample .java 代码如下：

```
package com.ThemeExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ThemeExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //加载主题 CustomTheme1,Theme 定义在 strings.xml 中
```

```
        setTheme(R.style.CustomTheme1);
        setContentView(R.layout.main);
    }
}
```

res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Theme 示例" />
</LinearLayout>
```

res/values/strings.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">ThemeExample</string>
    <!-- Theme 主题 -->
    <style name="CustomTheme1">
        <item name="android:windowNoTitle">false</item>
        <item name="android:textColor">#FFFF0000</item>
        <item name="android:background">#FFF0F0F0</item>
        <item name="android:textSize">18sp</item>
    </style>
</resources>
```

运行效果如图 4.52 所示。

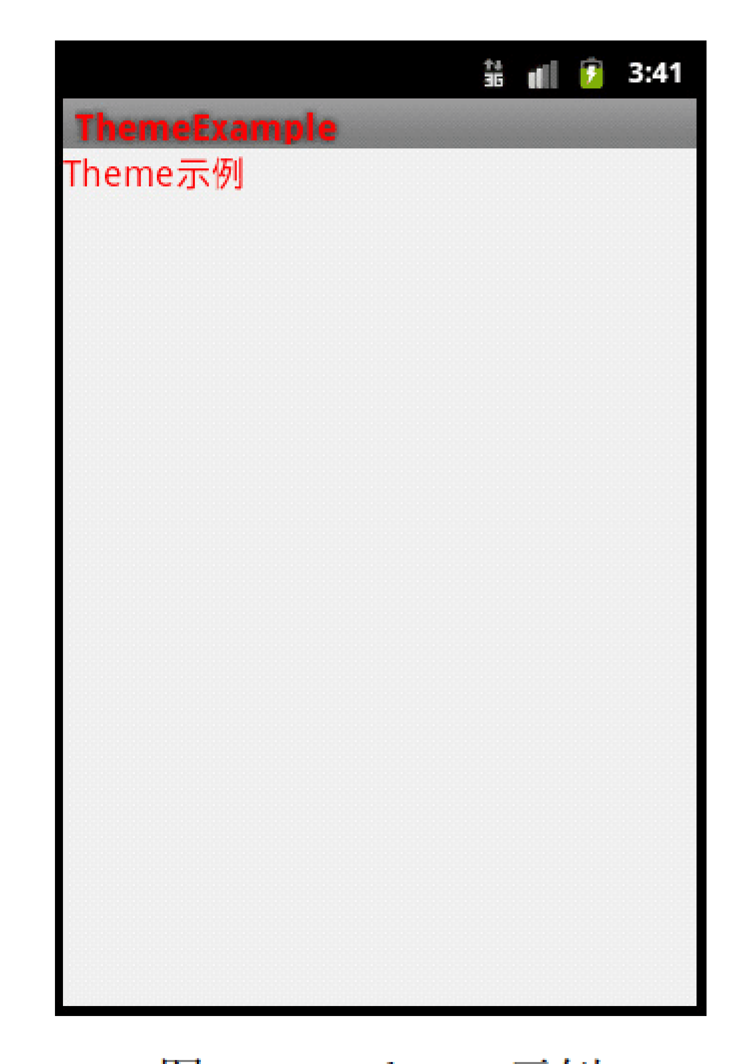


图 4.52　Theme 示例

4.27　监听屏幕旋转——onConfigurationChanged

在 Android 中屏幕发生了旋转（横向，纵向切换），是件很郁闷的事情，当前的 Activity 会被销毁，然后重新创建一个新屏幕方向的 Activity，有时候应用的参数很多，一般不会考虑两种屏幕的情况，所以需要禁用屏幕旋转功能，目前大多数的游戏都禁用了该功能。

我们可以通过在 AndroidManifest.xml 中指定 Activity 的 android:screenOrientation 属性来限制当前 Activity 显示的方式，其中 android:screenOrientation="landscape"表示横向显示，Activity 设置了该值后，始终横向显示。android:screenOrientation="portrait"表示纵向显示。可以通过重写 Activity 中的 onConfigurationChanged()方法来监听屏幕旋转事件，如果我们想让 Activity 捕获 onConfigurationChanged 事件，必须做两件事情：

第一声明权限，在 AndroidManifest.xml 中添加下面权限：

```
<uses-permission
android:name="android.permission.CHANGE_CONFIGURATION"></uses-permission>
```

第二声明 Activity 要捕获的事件类型，通过设置 Activity 属性 android:configChanges="orientation|keyboard"实现，多个事件用"|"分开，例如，下面的示例中对横屏及竖屏事件进行了捕获。

ScreenOrientationExample .java 代码如下：

```
package com.ScreenOrientationExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ScreenOrientationExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);//加载布局资源文件
    }
    //屏幕旋转时触发该方法
    @Override
    public void onConfigurationChanged(Configuration newConfig) {
        // TODO Auto-generated method stub
        //newConfig.orientation 获取当前屏幕状态是横向或者纵向，Configuration.
        ORIENTATION_PORTRAIT：表示竖屏
        //Configuration.ORIENTATION_LANDSCAPE:表示横屏
        if (newConfig.orientation == Configuration.ORIENTATION_PORTRAIT) {
            Toast.makeText(ScreenOrientationExample.this, "现在是竖屏",
            Toast.LENGTH LONG).show();
        }

        if (newConfig.orientation == Configuration.ORIENTATION LANDSCAPE) {
            Toast.makeText(ScreenOrientationExample.this, "现在是横屏",
            Toast.LENGTH_LONG).show();
        }
        super.onConfigurationChanged(newConfig);
    }
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#FFFFFFFF">
    <TextView android:layout_width="fill_parent" android:textColor="#FF
    000000"
        android:layout_height="wrap_content" android:text="屏幕旋转事件监听" />
</LinearLayout>
```

AndroidManifest.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ScreenOrientationExample" android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/
    app name">
        <activity android:name=".ScreenOrientationExample"
            android:label="@string/app name" android:configChanges="ori-
            entation|keyboard">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!-- 屏幕旋转事件权限 -->
        <uses-permission
android:name="android.permission.CHANGE CONFIGURATION"></uses-permission>
    </application>
</manifest>
```

运行效果如图 4.53 和图 4.54 所示。

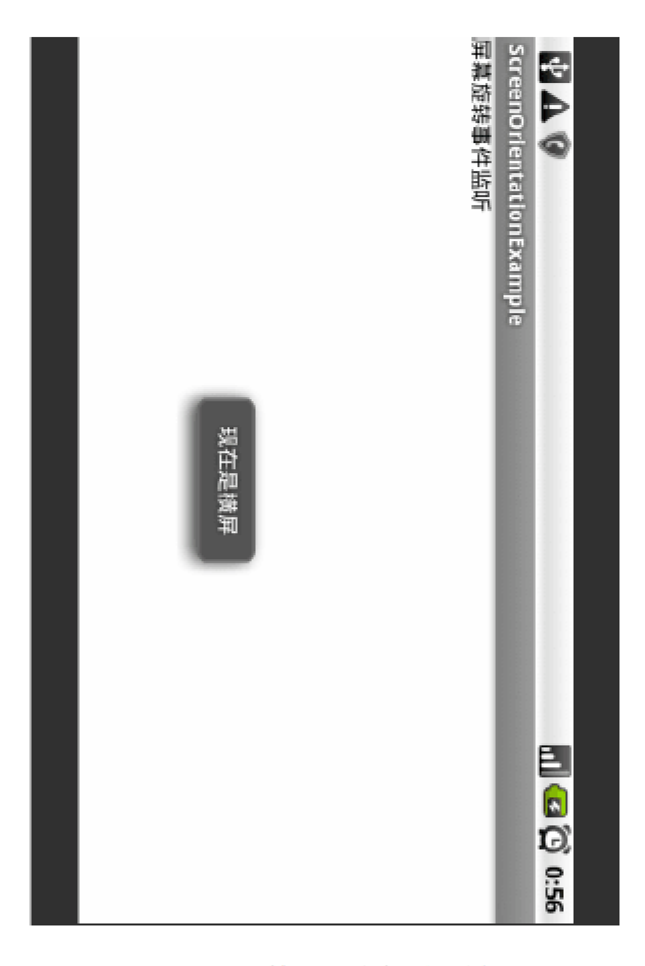


图 4.53　屏幕旋转事件效果 1



图 4.54　屏幕旋转事件效果 2

4.28　监听长时单击——OnLongClickListener

OnLongClickListener 接口用来监听 View 的长按事件，用户按住该元素或者按住轨迹球时调用该事件，该接口对应的回调方法原型为：

```
public boolean onLongClick(View v)
```

参数 v：为事件源组件，即长时间按下此组件才会触发该方法。

返回值：返回 true 时，表示已经完整的处理了该事件，并不希望其他回调方法再次进行处理；返回 false 时，表示没有完全处理完该事件，希望其他方法继续对其进行处理。下面通过例子了解 OnLongClickListener 事件的使用。

LongClickListenerExample .java 代码如下：

```
package com.LongClickListenerExample;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class LongClickListenerExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);          //加载布局资源文件

        Button longBut = (Button) this.findViewById(R.id.longBut);
                                                //从资源文件中获取 Button
        longBut.setOnLongClickListener(new OnLongClickListener()
        {// 监听长时单击时间
                    @Override
                    public boolean onLongClick(View v) {
                        // TODO Auto-generated method stub
                        //
                        Toast.makeText(LongClickListenerExample.this,
                                "OnLongClickListener 事件", Toast.LENGTH_LONG)
                                .show();
                        return false;
                    }

                });
    }
}
```

Res/layout/main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLayout" android:orientation="vertical"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:background="#FFFFFFFF">
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="长时单击事件"
        android:textColor="#FF000000" android:textSize="18px" />
    <Button android:id="@+id/longBut" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="长按我噢" />
</LinearLayout>
```

运行结果如图 4.55 所示。

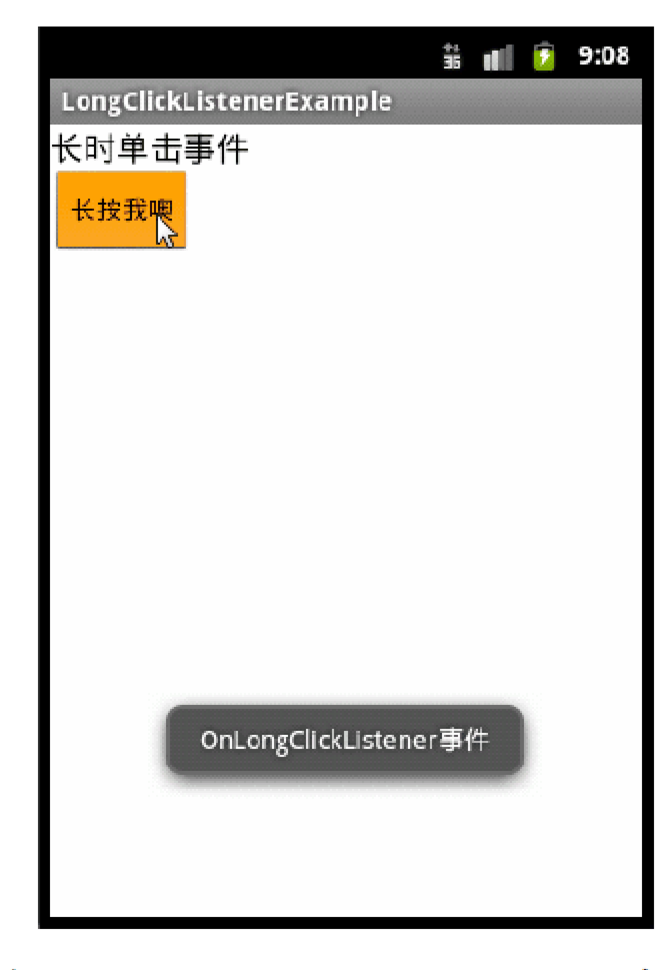


图 4.55　OnLongClickListener 事件

第 5 章　手机硬件设备的使用

本章将对 Android 手机硬件设备做一个详细的讲解，具体地说，就是多媒体、摄像头、短信、拨号、网络、蓝牙等一系列的介绍与应用。本章涉及的知识点较多，这些应用涉及比较广的方面，要开发一些实用软件，那么这章的知识就必不可少。

5.1　使用媒体 API

Android 中的多媒体架构基于第三方 PacketVideo 公司的 OpenCore 来实现，支持所有通用的音频、视频、静态图像格式。Android 平台的音视频采集，播放的操作都是通过 OpenCore 来实现，OpenCore 是 Android 多媒体框架的核心。

OpenCore 多媒体框架有一套通用可扩展的接口针对第三方的多媒体编解码器、输入、输出设备等等，支持多媒体文件的播放、下载，下面就来看如何具体的操作多媒体应用。

5.1.1　从源文件中播放

Android 的多媒体，主要学习重点是 MediaPlayer 对象的操作，使用 MediaPlayer.create()方法来创建播放器播放资源；使用 MediaPlayer.start()方法来开始播放；使用 MediaPlayer.pause()方法来暂停播放；使用 MediaPlayer.stop()方法来停止播放。

在这个示例中，我们用了 3 个按钮，它们的功能分别是开始播放（MediaPlayer.start()）、暂停播放（MediaPlayer.pause()）和停止播放（MediaPlayer.stop()）。当单击播放按钮时，程序会从指定的手机资源中取得 dudong.mp3 文件并开始播放，单击暂停按钮，文件暂停播放；再次单击，取消文件暂停并开始播放；单击停止按钮，文件停止播放。

我们先在项目的 res 文件夹下，添加一个名为“raw”的文件夹，再将 dudong.mp3 文件导入到新建的 res/raw 文件夹里，如图 5.1 所示。

现在准备工作就做好了，那让我们来看一下运行后的效果，如图 5.2 所示。

Player.java 代码如下：

```
package com.player;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class Player extends Activity {
    private MediaPlayer mediaPlayer = null;     //创建一个空 MediaPlayer 对象
    private Button startButton = null;          //播放 Button 组件对象
    private Button pauseButton = null;          //暂停 Button 组件对象
    private Button stopButton = null;           //停止 Button 组件对象
    private TextView nameTextView = null;       //文件名称 TextView 组件对象
```

```
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        nameTextView = (TextView) findViewById(R.id.mp3 name);
                                            //实例化文件名称 TextView 组件对象
        nameTextView.setText("dudong.mp3");//设置文件名称
        startButton = (Button) findViewById(R.id.button start);
                                            //实例化播放 Button 组件对象
        startButton.setOnClickListener(new Button.OnClickListener() {
                                            //添加播放按钮单击事件监听
                    @Override
                    public void onClick(View arg0) {
                        start();            //调用 MP3 播放方法
                    }
                });
        pauseButton = (Button) findViewById(R.id.button_pause);
                                            //实例化暂停 Button 组件对象
        pauseButton.setOnClickListener(new Button.OnClickListener() {
                                            //添加暂停按钮单击事件监听
                    @Override
                    public void onClick(View arg0) {
                        pause();            //调用 MP3 暂停播放方法
                    }
                });
        stopButton = (Button) findViewById(R.id.button stop);
                                            //实例化停止 Button 组件对象
        stopButton.setOnClickListener(new Button.OnClickListener() {
                                            //添加停止按钮单击事件监听
                    @Override
                    public void onClick(View arg0) {
                        stop();             //调用 MP3 停止播放方法
                    }
                });
    }

    /**
     * MP3 开始播放方法
     */
    public void start() {
        try {
            if (mediaPlayer != null) {          //判断 MediaPlayer 对象不为空
                if (mediaPlayer.isPlaying()) { //判断 MediaPlayer 对象正在播
                放中，并不执行以下程序
                    return;
                }
            }
            stop();                             //调用停止播放方法
            mediaPlayer = MediaPlayer.create(this, R.raw.dudong);
                                                //加载资源文件里的 MP3 文件
            //文件播放完毕监听事件
            mediaPlayer
                    .setOnCompletionListener(new MediaPlayer.OnCompl-
                    Listener() {
                        @Override
                        public void onCompletion(MediaPlayer arg0) {
                                                //覆盖文件播出完毕事件
```

```
                            //解除资源与 MediaPlayer 的赋值关系，让资源可以为其他
                            程序利用
                            mediaPlayer.release();
                            startButton.setText("播放");
                        }
                    });
            //文件播放错误监听
            mediaPlayer.setOnErrorListener(new MediaPlayer.OnErrorListener() {
                @Override
                public boolean onError(MediaPlayer arg0,int arg1, int arg2) {
                    //解除资源与 MediaPlayer 的赋值关系，让资源可以为其他程序利用
                    mediaPlayer.release();
                    return false;
                }
            });
            mediaPlayer.start();                    //开始播放
            startButton.setText("正在播放");
            pauseButton.setText("暂停");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    /**
     * MP3 播放暂停方法
     */
    public void pause() {
        try {
            if (mediaPlayer != null) {          //判断 MediaPlayer 对象不为空
                if (mediaPlayer.isPlaying()) {//判断 MediaPlayer 对象正在播放中
                    mediaPlayer.pause();        //暂停播放
                    pauseButton.setText("取消暂停");
                } else {
                    mediaPlayer.start();        //开始播放
                    pauseButton.setText("暂停");
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * MP3 停止播放方法
     */
    public void stop() {
        try {
            if (mediaPlayer != null) {          //判断 MediaPlayer 对象不为空
                mediaPlayer.stop();             //停止播放
                startButton.setText("播放");
                pauseButton.setText("暂停");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

以上就是这个示例的完整代码。需要注意的是，MediaPlayer 设置 OnCompletionListener()与 OnErrorListener()事件，用以处理播放结束与发生错误的事件处理，在播放结束或者发生错误时，都必须调用 MediaPlayer.release()方法将相关文件与资源释放出来，以避免 MediaPlayer 的资源占用。

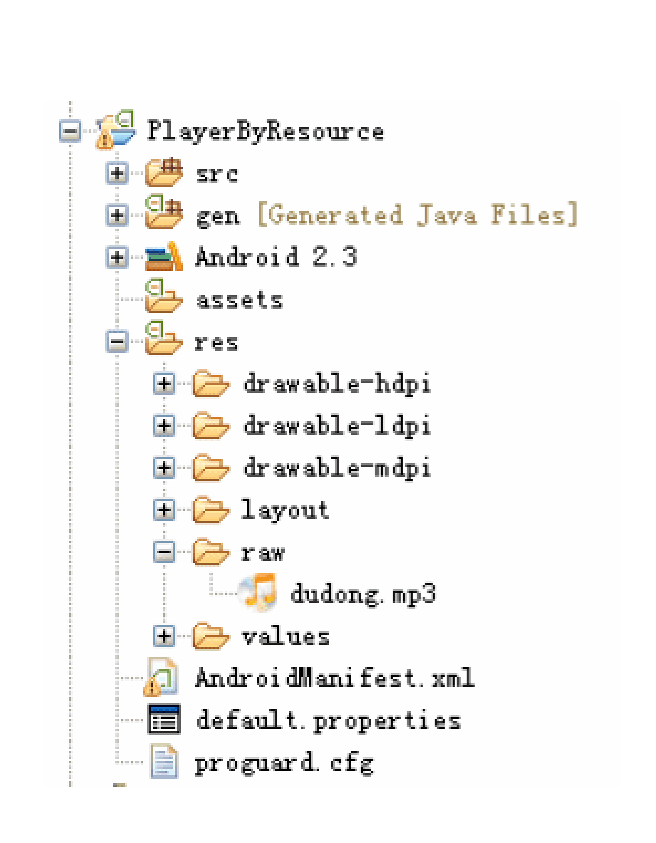


图 5.1 “raw”文件夹结构图



图 5.2　运行效果图

5.1.2　从文件系统中播放

这一节主要讲解 MediaPlayer 对象加载外部 MP3 媒体文件的方式。要加载外部媒体文件其实并不难，主要通过 MediaPlayer.setDataSource()方法来实现。构建 setDataSource()的方法有很多，比较简单的方法就是直接传入 MP3 媒体文件的路径。

在这个示例中，我们用了 3 个按钮，它们的功能分别是开始播放（MediaPlayer.start()）、暂停播放（MediaPlayer.pause()）和停止播放（MediaPlayer.stop()），当单击播放按钮时，程序会从存储卡（SDCard）中指定位置取得 dudong.mp3 文件并开始播放。单击暂停按钮，文件暂停播放；再次单击，取消文件暂停并开始播放；单击停止按钮，文件停止播放。



图 5.3　运行效果图

我们先把 dudong.mp3 媒体文件导入到至存储卡（SDCard）中，现在准备工作就做好了，那让我们来看一下运行后的效果，如图 5.3 所示。

实现上述效果的代码如下：

Player.java

```
package com.player;
```

```
import java.io.File;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class Player extends Activity {
    private MediaPlayer mediaPlayer = null;    //创建一个空 MediaPlayer 对象
    private Button startButton = null;         //播放 Button 组件对象
    private Button pauseButton = null;         //暂停 Button 组件对象
    private Button stopButton = null;          //停止 Button 组件对象
    private TextView nameTextView = null;      //文件名称 TextView 组件对象
    private boolean isPause = false;           //是否暂停
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        nameTextView = (TextView) findViewById(R.id.mp3_name);
                                          //实例化文件名称 TextView 组件对象
        nameTextView.setText("dudong.mp3");    //设置文件名称
        startButton = (Button) findViewById(R.id.button_start);
                                               //实例化播放 Button 组件对象
        startButton.setOnClickListener(new Button.OnClickListener() {
                                               //添加播放按钮单击事件监听
                    @Override
                    public void onClick(View arg0) {
                        start();               //调用 MP3 播放方法
                    }
                });
        pauseButton = (Button) findViewById(R.id.button_pause);
                                               //实例化暂停 Button 组件对象
        pauseButton.setOnClickListener(new Button.OnClickListener() {
                                               //添加暂停按钮单击事件监听

                    @Override
                    public void onClick(View arg0) {
                        pause();               //调用 MP3 暂停播放方法
                    }
                });
        stopButton = (Button) findViewById(R.id.button_stop);
                                               //实例化停止 Button 组件对象
        stopButton.setOnClickListener(new Button.OnClickListener() {
                                               //添加停止按钮单击事件监听
                    @Override
                    public void onClick(View arg0) {
                        stop();                //调用 MP3 停止播放方法
                    }
                });
    }

    /**
     * MP3 开始播放方法
     */
    public void start() {
        try {
            if (mediaPlayer != null) {         //判断 MediaPlayer 对象不为空
                if (mediaPlayer.isPlaying()) {//判断 MediaPlayer 对象正在播
                放中，并不执行以下程序
                    return;
```

```
                }
            }
            if (isPause) {// 判断 MediaPlayer 对象是否暂停，如果暂停就不重新播放
                return;
            }
            mediaPlayer = new MediaPlayer();
            //文件播放完毕监听事件
            mediaPlayer
                    .setOnCompletionListener(new MediaPlayer.OnCompLeti-
                    onListener() {
                        @Override
                        public void onCompletion(MediaPlayer arg0) {
                                                  //覆盖文件播出完毕事件
                            //解除资源与 MediaPlayer 的赋值关系，让资源可以为其
                            他程序利用
                            mediaPlayer.release();
                            startButton.setText("播放");
                            isPause = false;        //取消暂停状态
                            mediaPlayer=null;
                        }
                    });
            //文件播放错误监听
            mediaPlayer.setOnErrorListener(new MediaPlayer.OnErrorListener() {

                @Override
                public boolean onError(MediaPlayer arg0, int arg1, int arg2) {
                    //解除资源与 MediaPlayer 的赋值关系，让资源可以为其他程序利用
                    mediaPlayer.release();
                    isPause = false;            //取消暂停状态
                    mediaPlayer=null;
                    return false;
                }
            });
            String sdCard = Environment.getExternalStorageDirectory().getPath();
            mediaPlayer.setDataSource(sdCard + File.separator + "dudong.
            mp3");                              //为 MediaPlayer 设置数据源
            mediaPlayer.prepare();              //准备播放
            mediaPlayer.start();                //开始播放
            startButton.setText("正在播放");
            pauseButton.setText("暂停");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * MP3 播放暂停方法
     */
    public void pause() {
        try {
            if (mediaPlayer != null) {          //判断 MediaPlayer 对象不为空
                if (mediaPlayer.isPlaying()) {//判断 MediaPlayer 对象正在播放中
                    mediaPlayer.pause();        //暂停播放
                    pauseButton.setText("取消暂停");
                    isPause = true;             //暂停状态
                } else {
                    mediaPlayer.start();        //开始播放
```

```
                        pauseButton.setText("暂停");
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        /**
         * MP3 停止播放方法
         */
        public void stop() {
            try {
                if (mediaPlayer != null) {          //判断 MediaPlayer 对象不为空
                    mediaPlayer.stop();             //停止播放
                    startButton.setText("播放");
                    pauseButton.setText("暂停");
                    isPause = false;                //取消暂停状态
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

以上就是这个示例的完整代码，重点 MediaPlayer 对象的 MediaPlayer.setDataSource()方法来加载媒体文件，MediaPlayer.setDataSource()的构造方法有如下 4 种方式。

- void setDataSource(String path) ：传入文件路径或网址 URL，也可以是 rtsp://流文件的 URL 地址。
- void setDataSource(FileDescriptor fd) ：在未知的 FileDescriptor 对象的数据长度之下，可以仅传入 fd 即可，由 Android 直接从头开始播放。
- void setDataSource(FileDescriptor fd,long offset,long length) ：以传入 FileDescriptor 对象作为播放来源，并传入 offset 的片段开始播放，以及 FileDescriptor 的对象数据长度。
- void setDataSource(Context context,Uri uri) ：传入 Uri 对象的方式，通常需要使用 Uri.parse()的方式，解析手机里的 Context 对象。

5.1.3　从网络中播放

随着 3G 技术的逐渐成熟，已经步入了移动互联网的时代，资费不断降低，直接利用网络资源已经不再是问题，这一节就让我们来揭开如何通过网络来播放媒体文件这一神秘面纱。要通过网络来播放媒体文件，比较简单的方法就是通过 MediaPlayer.setDataSource()方法，直接传入网络媒体资源文件的地址来实现。

在这个示例中，我们用了 3 个按钮，它们的功能分别是开始播放（MediaPlayer.start()）、暂停播放（MediaPlayer.pause()）和停止播放（MediaPlayer.stop()），当单击播放按钮时，程序会通过媒体文件的网络地址，获取媒体文件资源并开始播放。单击暂停按钮，文件暂停播放；再次单击，取消文件暂停并开始播放；单击停止按钮，文件停止播放；我们来看

一下运行后的效果，如图 5.4 所示。



图 5.4　运行效果图

Player.java 代码如下：

```
package com.player;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class Player extends Activity {
    private MediaPlayer mediaPlayer = null;//创建一个空 MediaPlayer 对象
    private Button startButton = null;      //播放 Button 组件对象
    private Button pauseButton = null;      //暂停 Button 组件对象
    private Button stopButton = null;       //停止 Button 组件对象
    private TextView nameTextView = null;   //文件名称 TextView 组件对象
    private boolean isPause = false;        //是否暂停
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        nameTextView = (TextView) findViewById(R.id.mp3_name);// 实例化文
        件名称 TextView 组件对象
    nameTextView.setText("http://5.gaosu.com/download/ring/000/086/8e2c2
    3ec414592efdfad5353c21ce888.mp3");      //设置文件名称
        startButton = (Button) findViewById(R.id.button_start);
                                            //实例化播放 Button 组件对象
        startButton.setOnClickListener(new Button.OnClickListener() {
                                            //添加播放按钮单击事件监听
                    @Override
                    public void onClick(View arg0) {
                        start();            //调用 MP3 播放方法
                    }
                });
        pauseButton = (Button) findViewById(R.id.button_pause);
                                            //实例化暂停 Button 组件对象
        pauseButton.setOnClickListener(new Button.OnClickListener() {
```

```
                                              //添加暂停按钮单击事件监听
                    @Override
                    public void onClick(View arg0) {
                        pause();              //调用 MP3 暂停播放方法
                    }
                });
        stopButton = (Button) findViewById(R.id.button stop);
                                              //实例化停止 Button 组件对象
        stopButton.setOnClickListener(new Button.OnClickListener() {
                                              //添加停止按钮单击事件监听
                    @Override
                    public void onClick(View arg0) {
                        stop();               //调用 MP3 停止播放方法
                    }
                });
    }
    /**
     * MP3 开始播放方法
     */
    public void start() {
        try {
            if (mediaPlayer != null) {        //判断 MediaPlayer 对象不为空
                if (mediaPlayer.isPlaying()) {//判断 MediaPlayer 对象正在播
                放中，并不执行以下程序
                    return;
                }
            }
            if (isPause) {//判断 MediaPlayer 对象是否暂停，如果暂停就不重新播放
                return;
            }
            mediaPlayer = new MediaPlayer();
            //文件播放完毕监听事件
            mediaPlayer
                    .setOnCompletionListener(new MediaPlayer.OnCompletio-
                    nListener() {
                        @Override
                        public void onCompletion(MediaPlayer arg0) {
                                                  //覆盖文件播出完毕事件
                            //解除资源与 MediaPlayer 的赋值关系，让资源可以为其
                            他程序利用
                            mediaPlayer.release();
                            startButton.setText("播放");
                            isPause = false;      //取消暂停状态
                            mediaPlayer = null;
                        }
                    });
            //文件播放错误监听
            mediaPlayer.setOnErrorListener(new MediaPlayer.OnErrorListener()
{

                @Override
                public boolean onError(MediaPlayer arg0, int arg1, int arg2) {
                    //解除资源与 MediaPlayer 的赋值关系，让资源可以为其他程序利用
                    mediaPlayer.release();
                    isPause = false;              //取消暂停状态
                    mediaPlayer = null;
                    return false;
                }
```

```
            });
            //MP3 网络资源
        String path = "http://5.gaosu.com/download/ring/000/086/8e2c23ec-
        414592efdfad5353c21ce888.mp3";
            mediaPlayer.setDataSource(path);    //为 MediaPlayer 设置数据源
            mediaPlayer.prepare();              //准备播放
            mediaPlayer.start();                //开始播放
            startButton.setText("正在播放");
            pauseButton.setText("暂停");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * MP3 播放暂停方法
     */
    public void pause() {
        try {
            if (mediaPlayer != null) {      //判断 MediaPlayer 对象不为空
                if (mediaPlayer.isPlaying()) {//判断 MediaPlayer 对象正在播放中
                    mediaPlayer.pause();    //暂停播放
                    pauseButton.setText("取消暂停");
                    isPause = true;         //暂停状态
                } else {
                    mediaPlayer.start();    //开始播放
                    pauseButton.setText("暂停");
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * MP3 停止播放方法
     */
    public void stop() {
        try {
            if (mediaPlayer != null) {      //判断 MediaPlayer 对象不为空
                mediaPlayer.stop();         //停止播放
                startButton.setText("播放");
                pauseButton.setText("暂停");
                isPause = false;            //取消暂停状态
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

以上就是这个示例的完整代码。MediaPlayer 对象加载网络媒体资源文件除了上述加载方法，还可以通过 MediaPlayer.create()方法来加载，由于这种方法不常用，在这里我们做一个简单的介绍，代码如下：

```
        //MP3 网络资源路径
```

```
String path = "http://5.gaosu.com/download/ring/000/086/8e2c23ec
414592efdfad5353c21ce888.mp3";
Uri uri=Uri.parse(path);                    //实例化 Uri
MediaPlayer mediaPlayer=MediaPlayer.create(this, uri);
                                            //实例化 MediaPlayer
mediaPlayer.start();                        //开始播放
mediaPlayer.pause();                        //暂停
mediaPlayer.stop();                         //停止
```

5.1.4　录制多媒体

Android 系统提供了对音频及视频的支持，当然这需要手机本身的硬件支持，现在大部分手机都支持，Android 中的 MediaRecorder 类提供了相关方法。

在这个示例中，我们用了两个按钮，它们的功能分别是开始录音（MediaRecorder.start()）和停止录音（MediaRecorder.stop()）。为了顺利且不限制录音时间，所以把录音文件存储到存储卡，当单击录音按钮时，程序先判断存储卡是否存在，如果存在开始录音；录音停止时程序将录音文件存储在存储卡中，同时下方的 ListView 中显示刚才录音完成的录音文件名称，单击录音文件名称打开播放程序，播放录音文件。我们来看一下运行后的效果，如图 5.5、图 5.6、图 5.7 和图 5.8 所示。



图 5.5　运行效果图 1



图 5.6　运行效果图 2



图 5.7　运行效果图 3



图 5.8　运行效果图 4

recording.java 代码如下：

```
package com.recording;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class Recording extends Activity {
    private Button startButton = null; //播放 Button 组件对象
    private Button stopButton = null;  //停止 Button 组件对象
    private ListView listView = null;  //用于显示文件列表的 ListView 组件对象
    private File[] files = null;       //File 数组
    private String dirPath = "";       //文件读/写指定目录
    private MediaRecorder mediaRecorder = null;
                                       //创建一个空 MediaRecorder 对象
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.recording);
        startButton = (Button) findViewById(R.id.button_start);
                                       //实例化播放 Button 组件对象
        stopButton = (Button) findViewById(R.id.button_stop);
                                       //实例化停止 Button 组件对象
        listView = (ListView) findViewById(R.id.listView);
                                       //实例化 ListView 组件对象
        stopButton.setEnabled(false);  //停止按钮失效
        setListViewData();             //为 ListView 填充数据
        startButton.setOnClickListener(new Button.OnClickListener(){
                                       //添加录音按钮单击事件监听

                @Override
                public void onClick(View arg0) {
                    startRecord();     //调用录音方法
                }
            });
        stopButton.setOnClickListener(new Button.OnClickListener(){
                                       //添加停止按钮单击事件监听

                @Override
                public void onClick(View arg0) {
                    stopRecord();      //调用停止录音方法
                }
            });
        listView.setOnItemClickListener(new OnItemClickListener(){
                                       //为 ListView 添加单击监听

                @Override
                public void onItemClick(AdapterView<?> arg0, View arg1,
                        int arg2, long arg3) {
                    Intent intent = new Intent();  //初始化 Intent
                    intent.setClass(Recording.this, Player.class);
                                                   //指定 intent 对象启动的类
                    intent.putExtra("filePath", files[arg2].getPath());
                                                   //函数传递
                    startActivity(intent);  //启动新的 Activity
                }
            });
    }

    /**
```

```
     * 为播放文件组件对象 ListView 填充数据
     */
    public void setListViewData() {
        ……//该处省略了具体实现代码，将在之后进行具体介绍
    }

    /**
     * 根据 File[]获取相应的集合
     *
     * @param files
     *            File 数组
     * @return List<HashMap<String, Object>>集合
     */
    public List<HashMap<String, Object>> getList(File[] files) {
            ……//该处省略了具体实现代码，将在之后进行具体介绍
    }

    /**
     * 构造适配器并为 ListView 添加适配器
     *
     * @param list
     *            HashMap 的集合
     * @param files
     *            File 数组
     */
    public void setAdapter(List<HashMap<String, Object>> list, File[] files) {
            ……//该处省略了具体实现代码，将在之后进行具体介绍
    }

    /**
     * 获取手机 SDCard 的存储状态
     *
     * @return 手机 SDCard 的存储状态(true/false)
     */
    public boolean getStorageState() {
        ……//该处省略了具体实现代码，将在之后进行具体介绍
    }

    /**
     * 开始录音方法
     */
    public void startRecord() {
        ……//该处省略了具体实现代码，将在之后进行具体介绍
    }

    /**
     * 停止录音方法
     */
    public void stopRecord() {
        ……//该处省略了具体实现代码，将在之后进行具体介绍
    }

    /**
     * 获取录音文件的路径
     *
     * @return
```

```
     */
    public String getRecordFilePath() {
        ……//该处省略了具体实现代码，将在之后进行具体介绍
    }
}
```

下面介绍手机开始录音方法，代码如下：

```
public void startRecord() {
        String path = getRecordFilePath();          //获取录音文件路径
        if (!"".equals(path)) {
            mediaRecorder = new MediaRecorder();    //实例化 MediaRecorder
            mediaRecorder.setAudioSource(MediaRecorder.AudioSource.DEFAULT);
                                                    //设置音频源
            mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);
                                                    //设置输出格式
            mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
                                                    //设置音频编码器
            mediaRecorder.setOutputFile(path);      //设置输出路径
            //文件录制错误监听
            mediaRecorder
                    .setOnErrorListener(new MediaRecorder.OnErrorListener() {

                        @Override
                        public void onError(MediaRecorder arg0, int arg1,
                                int arg2) {
                            if (mediaRecorder != null) {
                                //解除资源与 MediaRecorder 的赋值关系，让资源可
                                以为其他程序利用
                                mediaRecorder.release();
                            }
                        }
                    });
        }
        try {
            mediaRecorder.prepare();                //准备
            mediaRecorder.start();                  //开始录音
            startButton.setEnabled(false);          //录音按钮失效
            stopButton.setEnabled(true);            //停止按钮生效
            startButton.setText("录音中...");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

以上就是手机开始录音的方法。首先要获取录音文件路径，这里调用 getRecordFilePath()方法来获取录音文件的路径，具体这个方法怎么获取录音文件路径的，下面会详细讲解。

实例化 MediaRecorder 类，MediaRecorder 类包含了准备录音、开始录音等一系列方法。实例化后设置各种属性，如设置音频源 setAudioSource()方法、设置输出格式 setOutputFormat()方法等，这里我们全部都设置默认格式。值得注意的是，别忘了设置录音文件的路径 setOutputFile()方法。

为 MediaRecorder 类添加文件录制错误监听，当文件录音出现异常时，监听中的 onError()方法就会被调用。在 onError()方法中我们需要做的就是释放资源，判断 MediaRecorder 对象不为空，调用 MediaRecorder. release()方法解除资源与 MediaRecorder

的赋值关系，让资源可以为其他程序利用。

调用 MediaRecorder.start()方法来开始录音，但在这之前，要调用 MediaRecorder.prepare()方法来准备录音，然后用 Button.setEnabled(false)方法设置录音按钮失效，用 Button.setEnabled(true)方法设置停止按钮生效。

下面介绍获取录音文件的路径方法，代码如下：

```
public String getRecordFilePath() {
        String filePath = "";                         //声明文件路径
        boolean sdCardState = getStorageState();      //获取 SDCard 状态
        if (!sdCardState) { //判断 SDCard 状态是否为非正常状态
            return filePath;                          //返回空字符串路径
        }
        String sdCardPath = Environment.getExternalStorageDirectory().getPath();
                                                      //获取 SDCard 根目录路径
        File dirFile = new File(sdCardPath + File.separator + "recording");
                                              //自定义录音文件夹的 File 对象
        if (!dirFile.exists()) {                      //判断录音文件夹是否存在
            dirFile.mkdir();                          //创建文件夹
        }
        try {
            //创建一个前缀为 test 后缀为.amr 的录音文件，createTempFile 方法来创建
            是为了避免文件重复
            filePath = File.createTempFile("test", ".amr", dirFile)
                    .getAbsolutePath();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return filePath;                              //返回录音文件路径
    }
```

首先声明一个空字符串 filePath，用以存放录音文件路径，调用 getStorageState()方法来获取当前 SDCard 状态，判断 SDCard 状态是否为非正常状态，如果是就返回空字符串。

```
String sdCardPath =Environment.getExternalStorageDirectory().getPath();
```

以上这段代码，是用以获取当前 SDCard 根目录路径。拿到了 SDCard 根目录路径，在这个路径下，我们添加自己定义的录音文件存放路径，如在 SDCard 根目录下创建 recording 文件夹。当然你可以自己命名你的文件夹，用自定义录音文件存放路径实例化 File 对象，File. exists()方法判断文件路径是否存在，如果不存在，创建该路径对应的文件夹。

File.createTempFile()方法创建一个新的空录音文件，它有 3 个参数，第 1 个参数为要创建的文件的前缀字符串；第 2 个参数为要创建的文件的后缀字符串；第 3 个参数为要创建文件所在目录，getAbsolutePath()方法获取文件的绝对路径名字符串。

下面介绍停止录音方法，代码如下：

```
public void stopRecord() {
        if (mediaRecorder != null) {
            mediaRecorder.stop();                     //停止录音
            mediaRecorder.release();                  //释放资源
            startButton.setEnabled(true);             //录音按钮生效
            stopButton.setEnabled(false);             //停止按钮失效
            startButton.setText("录音");
            setListViewData();// 录音完成，重新为 ListView 填充数据
```

```
        }
    }
```

首先判断 MediaRecorder 对象是否为 NULL，如果为 NULL，就说明 MediaRecorder 对象没有实例化，那么我们就不需要任何操作；如果不为 NULL，MediaRecorder.stop()方法来停止录音，然后用 MediaRecorder.release()方法来释放资源，设置录音按钮生效、停止按钮失效，最后调用 setListViewData()方法重新为 ListView 填充数据。

下面介绍获取手机 SDCard 的存储状态方法，代码如下：

```
public boolean getStorageState() {
        if (Environment.getExternalStorageState().equals(
                Environment.MEDIA_MOUNTED)) {  //判断手机 SDCard 的存储状态
            return true;
        } else {
            AlertDialog alertDialog = new AlertDialog.Builder(this).create();
                                    //创建 AlertDialog 对象
            alertDialog.setTitle("提示信息");     //设置信息标题
            alertDialog.setMessage("未安装 SD 卡，请检查你的设备");
                                    //设置信息内容
            //设置确定按钮，并添加按钮监听事件
            alertDialog.setButton("确定", new OnClickListener() {
                @Override
                public void onClick(DialogInterface arg0, int arg1) {
                    // MainActivity.this.finish(); //结束应用程序
                }
            });
            alertDialog.show();                      //设置弹出提示框
            return false;
        }
    }
```

Environment.getExternalStorageState()方法是 Android 中用来获取手机 SDCard 的状态的方法。手机安装有 SDCard，并可以进行读写操作，那么 Environment.getExternalStorageState()方法返回的状态就等于 Environment.MEDIA_MOUNTED 这个常量的值，所以用 equals()方法来判断，如果相等就返回 true。

如果不相等，创建一个 AlertDialog 对象，AlertDialog 类是用以弹出提示消息的类；setTitle()方法设置错误信息标题；setMessage()方法设置具体信息内容；setButton()设置按钮，并为该按钮添加单击监听事件。当单击按钮时调用 finish()方法结束本应用程序，show()显示弹出框方法，这样弹出框才能显示到界面，最后返回 false。

下面介绍为播放文件组件对象 ListView 填充数据方法，代码如下：

```
    public void setListViewData() {
        boolean sdStatus = getStorageState();//调用获取手机 SDCard 的存储状态
        if (sdStatus) {//判断 SDCard 的存储状态，如果是 false，提示并结束本程序
            File sdCardFile = Environment.getExternalStorageDirectory();
                                    //获取 SDCard 根目录 File 对象
            dirPath = sdCardFile.getPath() + File.separator + "recording";
                                    //指定文件存放目录
            File dirFile = new File(dirPath);
            if (!dirFile.exists()) {            //判断文件存放目录是否存在
                dirFile.mkdir();                //创建文件存放目录
            }
            files = dirFile.listFiles();//获取文件存放目录中的文件 File 对象
```

```
            List<HashMap<String, Object>> list = getList(files);
                                        //调用获取相应的集合
            setAdapter(list, files); //调用构造适配器并为 ListView 添加适配器
        }
    }
```

首先调用 getStorageState()方法获取手机当前 SDCard 的存储状态，然后获取存放录音文件目录中的所有录音文件 File 对象；然后调用 getList()方法获取相应的集合，getList()方法将在后面做详细讲解；最后调用 setAdapter()方法为播放文件组件对象 ListView 添加适配器，setAdapter()方法也将在后面做详细讲解。

下面介绍根据 File[]获取相应的集合方法，代码如下：

```
    public List<HashMap<String, Object>> getList(File[] files) {
        List<HashMap<String, Object>> list = new ArrayList<HashMap<String,
        Object>>();// 创建 List 集合
        for (int i = 0; i < files.length; i++) {// 循环 File 数组
           HashMap<String, Object> hashMap = new HashMap<String, Object>();
                                        //创建 HashMap
            hashMap.put("file_name", files[i].getName());
                                        //往 HashMap 中添加文件名
            list.add(hashMap);          //将 HashMap 添加到 List 集合
        }
        return list;                    //返回 List 集合
    }
```

首先创建一个 HashMap 的 List 集合，也就是这个 List 集合中存放的是 HashMap，循环传过来的 File 数组，创建一个 HashMap，用 HashMap.put()方法添加文件名（files[i].getName()），file_name 为对应的 XML 布局文件中的控件名称。

下面介绍构造适配器并为播放文件组件对象 ListView 添加适配器方法，代码如下：

```
    public void setAdapter(List<HashMap<String, Object>> list, File[] files) {
        SimpleAdapter simpleAdapter = new SimpleAdapter(this, list,
                R.layout.file_list, new String[] { "file_name" },
                new int[] { R.id.fileName });      //实例化 SimpleAdapter

        listView.setAdapter(simpleAdapter);        //为 ListView 添加适配器
    }
```

首先实例化 SimpleAdapter（适配器）对象，参数 list 是根据 File[]获取相应的集合，R.layout.file_list 是我们自定义的布局文件，String[]取对应集合中的键，int[]取对应布局文件中的控件。然后用 ListView. setAdapter()方法将实例化的 SimpleAdapter 对象添加到播放文件组件对象 ListView 中。

Player.java

```
package com.recording;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class Player extends Activity {
……//该处省略了代码，代码内容实现类似 5.1.2 小节中的代码，读者可自行查阅随书光盘中的源代码
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.recording" android:versionCode="1" android:versionName
    ="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app
    name">
        <activity android:name=".Recording" android:label="@string/]app
        name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".Player"></activity>

    </application>
    <uses-sdk android:minSdkVersion="1" />
    <uses-permission android:name="android.permission.RECORD_AUDIO"></-
    uses-permission>
</manifest>
```

以上就是这个示例的完整代码，重点是 MediaRecorder 对象的操作。MediaRecorder.start()方法来开始录音；MediaRecorder.stop()方法来停止录音；MediaRecorder.setAudioSource()方法来设置音频源；MediaRecorder.setOutputFormat()方法来设置输出格式；MediaRecorder.setAudioEncoder()方法来设置音频编码器；MediaRecorder.setOutputFile()方法来设置输出路径。注意在调用 MediaRecorder.start()方法开始录音前，一定要调用 MediaRecorder.prepare()方法，在调用 MediaRecorder.stop()停止录音后和在错误监听事件 setOnErrorListener 里，一定要调用 MediaRecorder. release ()方法来释放资源，以便资源可以被其他程序利用。

特别需要注意的是，我们的程序是音频录制程序，在 Android 中我们就必须获取程序录制音频的权限，具体操作就是在 AndroidManifest.xml 中添加如下代码：

```
<uses-permission
android:name="android.permission.RECORD_AUDIO"></uses-permission>
```

5.2 使用摄像头

我们接着来介绍手机摄像头功能，手机拍照在现在已经很普遍了，Android 提供了相关的 API，下面就来介绍如何具体地操作媒体应用。

5.2.1 控制摄像头拍照

Android 系统提供了对摄像头拍照的支持，当然这需要手机本身的硬件支持，现在大部分手机都支持，Android 中的 Camera 类提供了相关方法。

在这个示例中，我们用了一个按钮，它的功能是开始预览（Camera.startPreview()），开始预览后，按下拍照键或者轨迹球进行拍照（Camera.takePicture()），程序将把拍照后的图片按预设好的图片大小和格式存储在手机存储卡（SDCard）里的自定义文件夹 MyCamera 里，程序在拍照后，预览照片 3 秒后将跳转到相机预览效果，可继续拍照。我

们来看一下运行后的效果，如图 5.9 和图 5.10 所示。



图 5.9　运行效果图

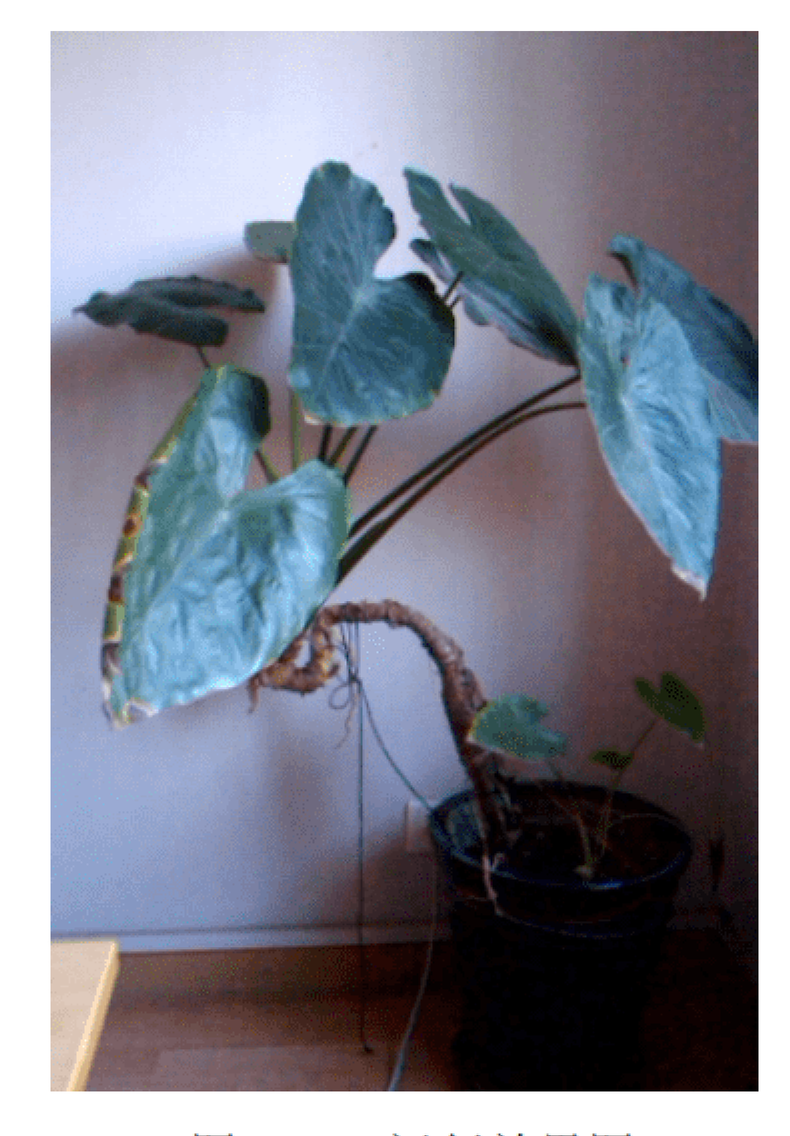


图 5.10　运行效果图

实现以上效果的代码如下：

CameraActivity.java

```
package com.mycamera;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class CameraActivity extends Activity implements SurfaceHolder.Cal-
lback {
    private SurfaceView surfaceView = null;//创建一个 SurfaceView 组件对象
    private SurfaceHolder surfaceHolder = null;
                                            //创建一个空 SurfaceHolder 对象
    private Camera camera = null;           //创建一个空 Camera 对象
    private boolean previewRunning = false;//预览状态
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getWindow().setFormat(PixelFormat.TRANSLUCENT);//窗口设置为半透明
        requestWindowFeature(Window.FEATURE_NO_TITLE); //窗口去掉标题
        getWindow().setFlags(WindowManager.LayoutParams.FLAG FULLSCREEN,
                WindowManager.LayoutParams.FLAG FULLSCREEN);
                                                        //窗口设置为全屏
        setRequestedOrientation(ActivityInfo.SCREEN ORIENTATION LANDSCAPE);
        setContentView(R.layout.camera);//调用 setRequestedOrientation 来
        翻转 Preview
        surfaceView = (SurfaceView) findViewById(R.id.surface camera);
                                            //实例化 SurfaceView 对象
        surfaceHolder = surfaceView.getHolder();  //获取 SurfaceHolder
        surfaceHolder.addCallback(this);          //注册实现好的 Callback
        surfaceHolder.setType(SurfaceHolder.SURFACE TYPE PUSH BUFFERS);
                                                  //设置缓存类型
    }

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
```

```
        //判断手机键盘按下的是否是拍照键、轨迹球键
        if (keyCode == KeyEvent.KEYCODE_CAMERA
                || keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {
            if (camera != null) {                  //判断 Camera 对象是否不为空
                //当按下相机按钮时，执行相机对象的 takePicture()方法，该方法有 3
                个回调对象做入参，不需要的时候可以设 null
                camera.takePicture(null, null, jpegCallback);
                changeByTime(3000);           //调用延迟方法，3 秒后重新预览拍照
            }
        }
        return super.onKeyDown(keyCode, event);
    }

    /**
     * 延迟方法
     *
     * @param time 毫秒
     */
    public void changeByTime(long time) {
        final Timer timer = new Timer();                //实例化 Timer 对象
        final Handler handler = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                switch (msg.what) {
                case 1:
                    stopCamera();                  //调用停止 Camera()方法
                    prepareCamera();               //调用初始化 Camera()方法
                    startCamera();                 //调用开始 Camera()方法
                    timer.cancel();                //撤销计时器
                    break;
                }
                super.handleMessage(msg);
            }
        };
        TimerTask task = new TimerTask() {
            @Override
            public void run() {
                Message message = new Message();
                message.what = 1;
                handler.sendMessage(message);
            }
        };
        timer.schedule(task, time);                //设定运行任务的时间
    }

    /**
     * 当预览界面的格式和大小发生改变时，该方法被调用
     */
    @Override
    public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int
    arg3) {
        startCamera();                             //调用开始 Camera()方法
    }

    /**
     * 初次实例化，预览界面被创建时，该方法被调用
     */
    @Override
```

```
    public void surfaceCreated(SurfaceHolder arg0) {
        prepareCamera();                          //调用初始化 Camera()方法
    }

    /**
     * 当预览界面被关闭时，该方法被调用
     */
    @Override
    public void surfaceDestroyed(SurfaceHolder arg0) {
        stopCamera();                             //调用停止 Camera()方法
    }

    /**
     * 初始化 Camera
     */
    public void prepareCamera() {
        camera = Camera.open();                           //初始化 Camera
        try {
            camera.setPreviewDisplay(surfaceHolder);      //设置预览
        } catch (IOException e) {
            camera.release();                             //释放相机资源
            camera = null;                                //置空 Camera 对象
        }
    }

    /**
     * 开始 Camera
     */
    public void startCamera() {
        if (previewRunning) {                             //判断预览开启
            camera.stopPreview();                         //停止预览
        }
        try {
            Camera.Parameters parameters = camera.getParameters();
                                                          //获得相机参数对象
            parameters.setPictureFormat(PixelFormat.JPEG); //设置格式
            //设置预览大小
            //parameters.setPreviewSize(480, 320);
            //设置自动对焦
            //parameters.setFocusMode("auto");
            //设置图片保存时的分辨率大小
            //parameters.setPictureSize(2048, 1536);
            camera.setParameters(parameters);  //给相机对象设置刚才设定的参数
            camera.setPreviewDisplay(surfaceHolder);
                        //设置用 SurfaceView 作为承载镜头取景画面的显示
            camera.startPreview();                //开始预览
            previewRunning = true;                //设置预览状态为 true
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * 停止 Camera
     */
    public void stopCamera() {
        if (camera != null) {                     //判断 Camera 对象不为空
```

```
            camera.stopPreview();                  //停止预览
            camera.release();                      //释放摄像头资源
            camera = null;                         //置空 Camera 对象
            previewRunning = false;                //设置预览状态为 false
        }
    }

    //照片拍摄之后的事件
    private PictureCallback jpegCallback = new PictureCallback() {

        @Override
        public void onPictureTaken(byte[] arg0, Camera arg1) {
            //获取存储卡(SDCard)的根目录
            String sdCard = Environment.getExternalStorageDirectory().getPath();
            //获取相片存放位置的目录
            String dirFilePath = sdCard + File.separator + "MyCamera";
            //获取当前时间的自定义字符串
            String date = (String) DateFormat.format(" yyyy-MM-dd hh-mm-ss",
                    new Date());
            //onPictureTaken 传入的第一个参数及为相片的 byte，实例化 Bitmap 对象
            Bitmap bitmap = BitmapFactory.decodeByteArray(arg0, 0, arg0.length);
            try {
                File dirFile = new File(dirFilePath);
                                                   //创建相片存放位置的 File 对象
                if (!dirFile.exists()) {           //判断路径是否不存在
                    dirFile.mkdir();               //创建该文件夹
                }
                //创建一个前缀为 photo，后缀为.jpg 的图片文件
                //createTempFile()方法来创建是为了避免文件重复
                File file = File.createTempFile("photo-", date + ".jpg",
                        dirFile);
                BufferedOutputStream bOutputStream = new BufferedOutputStream(
                        new FileOutputStream(file));
                //采用压缩文件的方法
                bitmap.compress(Bitmap.CompressFormat.JPEG, 80, bOutputStream);
                //清除缓存，更新 BufferedOutputStream
                bOutputStream.flush();
                //关闭 BufferedOutputStream
                bOutputStream.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    };
}
```

Welcome.java

```
package com.mycamera;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class Welcome extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.welcome);
        Button button = (Button) findViewById(R.id.camera_button);
                                                       //实例化 Button 组件对象
```

```
        button.setOnClickListener(new Button.OnClickListener() {
                                                    //为 Button 添加单击监听
                    @Override
                    public void onClick(View arg0) {
                        Intent intent = new Intent();   //初始化 Intent
                        intent.setClass(Welcome.this, CameraActivity.class);
                                                //指定 intent 对象启动的类
                        startActivity(intent); //启动新的 Activity
                    }
                });
    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mycamera" android:versionCode="1" android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/
    app_name">
        <activity android:name=".Welcome" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".CameraActivity"
            android:screenOrientation="portrait"></activity>
    </application>
    <uses-sdk android:minSdkVersion="1" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_SETTINGS" />
</manifest>
```

在 Android 手机系统平台中，拍照实现方式有两种，第一种方式是使用 android.hardware.Camera，第二种方式是使用 Intent("android.media.action.IMAGE_CAPTURE")。两种方式相对来说，各有各的优缺点，以上示例就是 android.hardware.Camera 实现方式，至于第二种实现方式就比较简单了，有兴趣的朋友可以去研究一下，这里我们就不做过多的介绍。

接下来，我们来具体讲解以上示例是如何实现摄像头拍照的。首先要想在自己的应用中使用摄像头，需要在 AndroidManifest.xml 中添加如下代码：

```
<uses-permission android:name="android.permission.CAMERA" />
```

然后我们要预览相机，就需要 SurfaceView 视图来显示我们的屏幕，需要在 camera.xml 中添加如下代码：

```
<SurfaceView android:id="@+id/surface_camera"
        android:layout_width="fill_parent" android:layout_height="fill_parent"
        android:layout_weight="1">
</SurfaceView>
```

CameraActivity 类需要实现（implements）用于接收摄像头预览界面变化信息的 SurfaceHolder.Callback 接口（interface），它实现了以下 3 个方法。

- surfaceChanged()：当预览界面的格式和大小发生改变时，该方法被调用。
- surfaceCreated()：初次实例化，预览界面被创建时，该方法被调用。

- surfaceDestroyed()：当预览界面被关闭时，该方法被调用。

5.2.2　控制摄像头摄像

这一节我们来学习如何控制摄像头摄像。Android 系统提供了对摄像头拍照及摄像的支持，Android 中的 MediaRecorder 类提供了相关方法。

在这个示例中，我们用了一个摄像按钮，它的功能是开始预览（Camera.startPreview()），开始预览后，进入视频录制的 Activity，在这个 Activity 中我们可以看到左侧的画面随着手机而变换，这就是预览效果。在这个 Activity 的右侧有两个按钮，一个录制按钮，一个停止按钮。单击录制按钮，程序开始录制视频（MediaRecorder.start()）；单击停止按钮，程序将结束录制（MediaRecorder.stop()），并且弹出提示框，提示视频已经录制完毕。是否保存。单击确定键，视频文件保存至 SDCard 中，视频录制结束，程序返回视频预览，我们可继续摄像。我们来看一下运行后的效果，如图 5.11、图 5.12、图 5.13 和图 5.14 所示。



图 5.11　运行效果图 1



图 5.12　运行效果图 2



图 5.13　运行效果图 3



图 5.14　运行效果图 4

实现以上效果的代码如下：

VideoRecording.java

```
package com.videorecording;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class VideoRecording extends Activity implements SurfaceHolder.Callback {
    private SurfaceView surfaceView = null;//创建一个空 SurfaceView 对象
```

```
    private SurfaceHolder surfaceHolder = null;
                                               //创建一个空 SurfaceHolder 对象
    private Button startButton = null; //创建开始录制按钮的 Button 组件对象
    private Button stopButton = null;  //创建停止录制按钮的 Button 组件对象
    private MediaRecorder mediaRecorder = null//创建一个空 MediaRecorder 对象
    private Camera camera = null;            //创建一个空 Camera 对象
    private boolean previewRunning = false;//预览状态
    private File videoFile = null;          //录制视频文件的 File 对象
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getWindow().setFormat(PixelFormat.TRANSLUCENT);//窗口设置为半透明
        requestWindowFeature(Window.FEATURE_NO_TITLE); //窗口去掉标题
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
                                                          //窗口设置为全屏
        //调用 setRequestedOrientation 来翻转 Preview
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        setContentView(R.layout.video);
        surfaceView = (SurfaceView) findViewById(R.id.surface_view);
                                                      //实例化 SurfaceView
        surfaceHolder = surfaceView.getHolder();      //获取 SurfaceHolder
        surfaceHolder.addCallback(this);              //注册实现好的 Callback
        surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
                                                      //设置缓存类型
        startButton = (Button) findViewById(R.id.start);
                                               //实例化开始录制按钮的 Button 组件对象
        stopButton = (Button) findViewById(R.id.stop);//实例化停止录制按钮
        的 Button 组件对象
        startButton.setEnabled(true);                         //摄像按钮生效
        stopButton.setEnabled(false);                         //停止按钮失效
        //添加摄像按钮单击事件监听
        startButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                startRecording();                             //调用开始摄像方法
            }
        });
        //添加停止按钮单击事件监听
        stopButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                stopRecording();                              //调用停止摄像方法
            }
        });
    }
    /**
     * 开始摄像方法
     */
    public void startRecording() {
        try {
            stopCamera();                                 //调用停止 Camera()方法
            if (!getStorageState()) {//判断是否有存储卡，如果没有就关闭页面
                VideoRecording.this.finish();         //结束应用程序
            }
```

```
            //获取存储卡(SDCard)的根目录
            String sdCard = Environment.getExternalStorageDirectory().getPath();
            //获取相片存放位置的目录
            String dirFilePath = sdCard + File.separator + "MyVideo";
            File dirFile = new File(dirFilePath);
                                            //获取录制文件夹的路径的 File 对象
            if (!dirFile.exists()) {        //判断文件夹是否存在
                dirFile.mkdir();            //创建文件夹
            }
            videoFile = File.createTempFile("video", ".3gp", dirFile);
                                            //创建录制视频临时文件
            mediaRecorder = new MediaRecorder();//初始化 MediaRecorder 对象
            mediaRecorder.setPreviewDisplay(surfaceHolder.getSurface());
                                            //预览
            mediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
                                            //视频源
            mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
                                            //录音源为麦克风
            //输出格式为 3gp
            mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.
            THREE_GPP);
            mediaRecorder.setVideoSize(480, 320);       //视频尺寸
            mediaRecorder.setVideoFrameRate(15);        //视频帧频率
            mediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.H263);
                                                        //视频编码
            mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
                                                        //音频编码
            mediaRecorder.setMaxDuration(10000);        //最大期限
            mediaRecorder.setOutputFile(videoFile.getAbsolutePath());
                                                        //保存路径
            mediaRecorder.prepare();                    //准备录制
            mediaRecorder.start();                      //开始录制
            //文件录制错误监听
            mediaRecorder
                    .setOnErrorListener(new MediaRecorder.OnErrorListener() {
                        @Override
                        public void onError(MediaRecorder arg0, int arg1,
                                int arg2) {
                            stopRecording();        //调用停止摄像方法
                        }
                    });
            startButton.setText("录制中");
            startButton.setEnabled(false);          //摄像按钮失效
            stopButton.setEnabled(true);            //停止按钮生效
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * 停止摄像方法
     */
    public void stopRecording() {
        if (mediaRecorder != null) {//判断 MediaRecorder 对象是否为空
            mediaRecorder.stop();                   //停止摄像
            mediaRecorder.release();                //释放资源
```

```
            mediaRecorder = null;                  //置空 MediaRecorder 对象
            startButton.setEnabled(true);          //摄像按钮生效
            stopButton.setEnabled(false);          //停止按钮失效
            startButton.setText("录制");
            isSave();                              //调用是否保存方法保存
        }
        stopCamera();                              //调用停止 Camera()方法
        prepareCamera();                           //调用初始化 Camera()方法
        startCamera();                             //调用开始 Camera()方法
    }

    /**
     * 初始化 Camera
     */
    public void prepareCamera() {
        camera = Camera.open();                            //初始化 Camera
        try {
            camera.setPreviewDisplay(surfaceHolder);       //设置预览
        } catch (IOException e) {
            camera.release();                              //释放相机资源
            camera = null;                                 //置空 Camera 对象
        }
    }

    /**
     * 开始 Camera
     */
    public void startCamera() {
        if (previewRunning) {                              //判断预览开启
            camera.stopPreview();                          //停止预览
        }
        try {
            //设置用 SurfaceView 作为承载镜头取景画面的显示
            camera.setPreviewDisplay(surfaceHolder);
            camera.startPreview();                     //开始预览
            previewRunning = true;                     //设置预览状态为 true
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * 停止 Camera
     */
    public void stopCamera() {
        if (camera != null) {                          //判断 Camera 对象不为空
            camera.stopPreview();                      //停止预览
            camera.release();                          //释放摄像头资源
            camera = null;                             //置空 Camera 对象
            previewRunning = false;                    //设置预览状态为 false
        }
    }

    /**
     * 手机按键监听事件
```

```
     */
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        //判断手机键盘按下的是否是返回键
        if (keyCode == KeyEvent.KEYCODE_BACK) {
            stopRecording();                        //调用停止摄像方法
            Intent intent = new Intent();           //初始化 Intent
            intent.setClass(VideoRecording.this, Welcome.class);
                                                    //指定 intent 对象启动的类
            intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                                                //清除该进程空间的所有 Activity
            startActivity(intent);              //启动新的 Activity
            VideoRecording.this.finish();       //销毁这个 Activity
        }
        return super.onKeyDown(keyCode, event);
    }

    /**
     * 是否保存录制的视频文件
     */
    public void isSave() {
        AlertDialog alertDialog = new AlertDialog.Builder(this).create();
                                                //创建 AlertDialog 对象
        alertDialog.setTitle("提示信息");       //设置信息标题
        alertDialog.setMessage("是否保存 " + videoFile.getName() + " 视频文
        件? ");                                 //设置信息内容
        //设置确定按钮，并添加按钮监听事件
        alertDialog.setButton("确定",
                new android.content.DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface arg0, int arg1) {
                    }
                });
        //设置取消按钮，并添加按钮监听事件
        alertDialog.setButton2("取消",
                new android.content.DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface arg0, int arg1) {
                        if (videoFile.exists()) {   //判断文件是否存在
                            videoFile.delete();     //删除该文件
                        }
                    }
                });
        alertDialog.show();                         //设置弹出提示框
    }
    /**
     * 获取手机 SDCard 的存储状态
     * @return 手机 SDCard 的存储状态(true/false)
     */
    public boolean getStorageState() {
        if (Environment.getExternalStorageState().equals(
                Environment.MEDIA_MOUNTED)) {//判断手机 SDCard 的存储状态
            return true;
        } else {
            AlertDialog alertDialog = new AlertDialog.Builder(this).create();
                                                    //创建 AlertDialog 对象
            alertDialog.setTitle("提示信息");       //设置信息标题
```

```
            alertDialog.setMessage("未安装 SD 卡，请检查你的设备");//设置信息内容
            //设置确定按钮，并添加按钮监听事件
            alertDialog.setButton("确定",
                    new android.content.DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface arg0, int arg1) {
                            VideoRecording.this.finish();   //结束应用程序
                        }
                    });
            alertDialog.show();                             //设置弹出提示框
            return false;
        }
    }

    /**
     * 当预览界面的格式和大小发生改变时，该方法被调用
     */
    @Override
    public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {
        startCamera();                              //调用开始 Camera()方法
    }

    /**
     * 初次实例化，预览界面被创建时，该方法被调用
     */
    @Override
    public void surfaceCreated(SurfaceHolder arg0) {
        prepareCamera();                            //调用初始化 Camera()方法
    }

    /**
     * 当预览界面被关闭时，该方法被调用
     */
    @Override
    public void surfaceDestroyed(SurfaceHolder arg0) {
        stopCamera();                               //调用停止 Camera()方法
    }
}
```

Welcome.java

```
package com.videorecording;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class Welcome extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.welcome);
        Button button = (Button) findViewById(R.id.camera_button);
                                                    //实例化 Button 组件对象
        button.setOnClickListener(new Button.OnClickListener() {
                                                    //为 Button 添加单击监听
                @Override
                public void onClick(View arg0) {
                    Intent intent = new Intent();  //初始化 Intent
                    intent.setClass(Welcome.this, VideoRecording.class);
                                                    //指定 intent 对象启动的类
                    startActivity(intent); //启动新的 Activity
```

```
                    }
                });
    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.videorecording" android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/-
    app name">
        <activity android:name=".Welcome" android:label="@string/app name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".VideoRecording"></activity>
    </application>
    <uses-sdk android:minSdkVersion="1" />
    <uses-permission android:name="android.permission.CAMERA"></uses-pe-
    rmission>
    <uses-permission android:name="android.permission.RECORD_AUDIO">}-
    </uses-permission>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_
    STORAGE"></uses-permission>

</manifest>
```

在 Android 手机平台中，摄像的例子不是很多，上面示例是一个完整的摄像程序，这里我们再来好好地分析一下以上示例。如果想要实现摄像头预览效果，首先要在自己的应用中使用摄像头，并且需要录制视频，还需要把录制好的视频文件存储在手机存储卡（SDCard），要完成这些操作，我们需要在 AndroidManifest.xml 中添加如下权限代码：

```
<uses-permission android:name="android.permission.CAMERA" ></uses-perm
ission>
<uses-permission android:name="android.permission.RECORD AUDIO"></uses-
permission>
<uses-permission android:name="android.permission.WRITE EXTERNAL STOR-
AGE"></uses-permission>
```

代码说明如下。

- ❑ android.permission.CAMERA：请求访问使用照相设备。
- ❑ android.permission.RECORD_AUDIO：允许程序录制音频。
- ❑ android.permission.WRITE_EXTERNAL_STORAGE：允许写入 External Storage 的相关权限。

我们要预览摄像机，就需要 SurfaceView 视图来显示我们的屏幕，那么需要在 video.xml 中添加如下代码：

```
<SurfaceView android:id="@+id/surface view"
        android:visibility="visible" android:layout width="400px"
        android:layout height="fill parent">
</SurfaceView>
```

在 Android 系统平台中，多媒体的录制是由 MeidaRecorder 类来完成的，它包括了 Audio 和 Video 的录制功能，主要设置方法如下。

- MediaRecorder.setPreviewDisplay()：设置视频录制预览。
- MediaRecorder.setVideoSource()：设置视频源。
- MediaRecorder.setAudioSource()：设置录音源。
- MediaRecorder.setOutputFormat()：设置输出格式。
- MediaRecorder.setVideoSize()：设置视频尺寸。
- MediaRecorder.setVideoFrameRate()：设置视频帧频率。
- MediaRecorder.setVideoEncoder()：设置视频编码。
- MediaRecorder.setAudioEncoder()：设置音频编码。
- MediaRecorder.setMaxDuration()：设置最大期限。
- MediaRecorder.setOutputFile()：设置保存路径。
- MediaRecorder.prepare()：准备录制。
- MediaRecorder.start()：开始录制。
- MediaRecorder.stop()：停止录制。
- MediaRecorder.release()：释放资源。

5.3　Android 电话功能

在 Android 手机系统平台中，要想让你的应用程序访问底层电话功能，要么创建自己的拨号器，要么启动一个新的拨号器。

Android 出于安全考虑，目前不允许你的应用程序直接打电话，而是调用 Android 内置打电话界面，当出现该界面，表示来电或者呼叫已经开始。本节我们将讲解如何打电话和监控电话状态，以及控制电话等功能。

在 Android 中实现打电话功能，最好的做法是创建自己的拨号器，使用 Intent 自动发起呼叫。当然你也可以用 Intent 启动一个拨号器应用，以下分别是打电话的两种 Activity Action。

- Intent.ACTION_CALL：自动发起呼叫，显示在通话中的应用。
- Intent.ACTION_DIAL：启动一个拨号器应用。

使用 Intent.ACTION_CALL 直接拨打电话，首先要在程序的 AndroidManifest.xml 文件中添加如下权限代码：

```
<uses-permission android:name="android.permission.CALL PHONE"></uses-p
ermission >
```

以上代码中，android.permission.CALL_PHONE 是指允许一个程序初始化一个电话拨号不需通过拨号用户界面要用户确认。

使用 Intent.ACTION_DIAL 启动一个拨号器应用，而不需要任何的权限就可以拨打电

话。如下是启动一个拨号器的代码：

```
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:10086"));
startActivity(intent);
```

在这个示例中，我们用了两个按钮和一个文本输入框，它们的功能分别是直接拨打电话、到拨打电话界面，以及输入电话号码。当我们输入完电话号码后，单击直接拨打电话按钮，程序将跳转至直接拨打页面，开始拨打电话；单击拨打电话界面，程序将跳转至拨打电话界面。我们来看一下运行后的效果，如图 5.15、图 5.16、图 5.17 和图 5.18 所示。



图 5.15　运行效果图 1



图 5.16　运行效果图 2

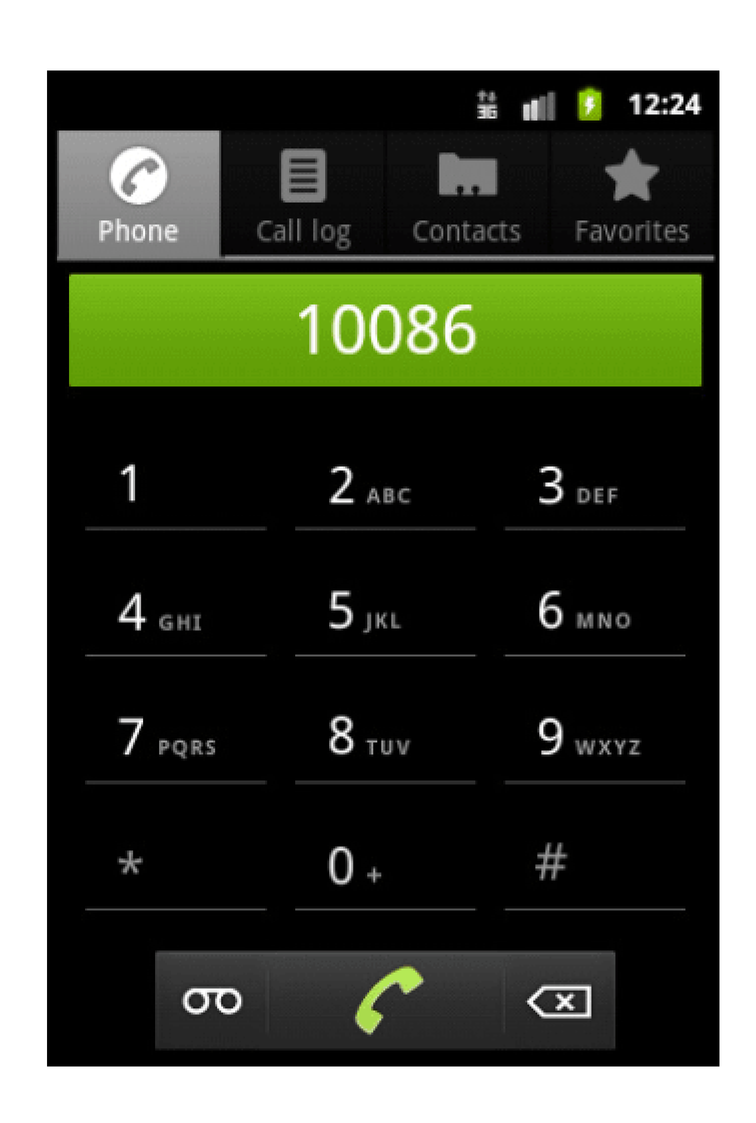


图 5.17　运行效果图 3

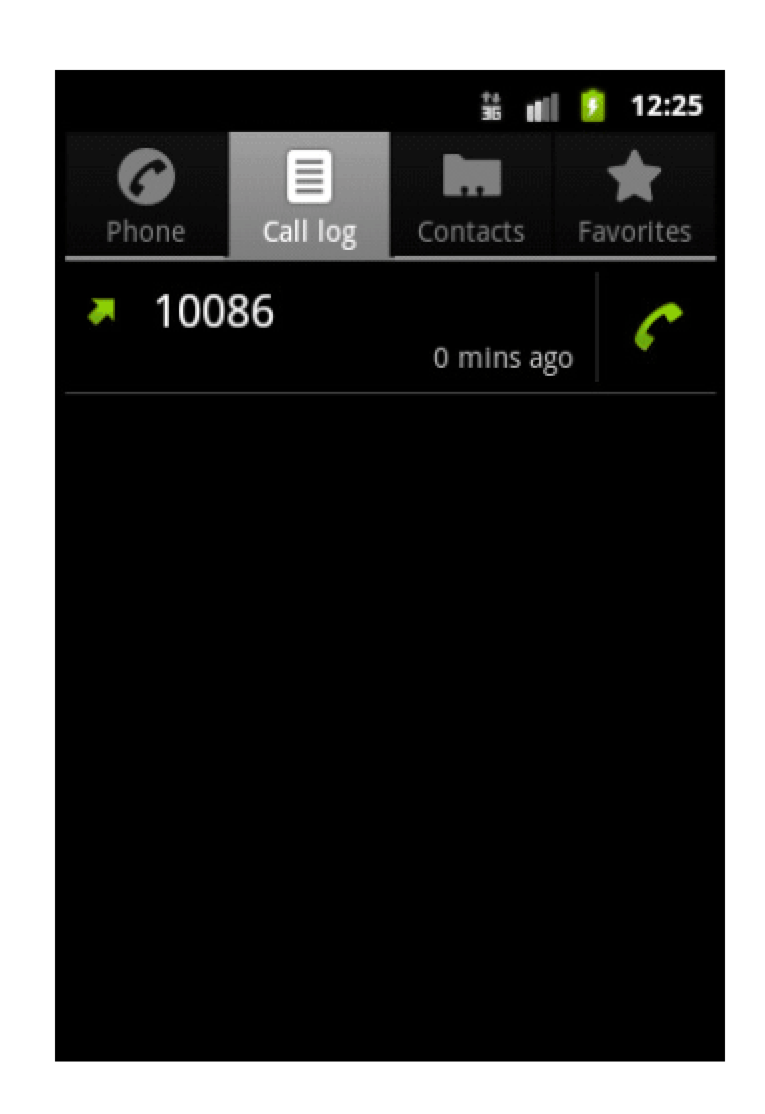


图 5.18　运行效果图 4

实现以上效果的代码如下：

CallActivity

```
package com.call;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class CallActivity extends Activity {
    private EditText editText = null;      //电话号码 EditText 组件对象
    private Button callButton = null;      //直接拨打按钮 Button 组件对象
    private Button dialButton = null;      //启动拨打界面按钮 Button 组件对象
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        editText = (EditText) findViewById(R.id.phone_number);
                             //实例化电话号码 EditText 组件对象
        callButton = (Button) findViewById(R.id.phone_call);
                             //实例化直接拨打按钮 Button 组件对象
        dialButton = (Button) findViewById(R.id.phone_dial);
                             //实例化启动拨打界面按钮 Button 组件对象
        callButton.setOnClickListener(new Button.OnClickListener()
                             {//添加 Button 按钮单击监听

            @Override
            public void onClick(View arg0) {
                call();              //调用直接打电话的方法
            }
        });
        dialButton.setOnClickListener(new Button.OnClickListener()
                             {//添加 Button 按钮单击监听
            @Override
            public void onClick(View arg0) {
                dial();              //调用启动一个拨号器的方法
            }
        });
    }

    /**
     * 直接打电话的方法
     */
    public void call() {
        String data = "tel:" + editText.getText();     //电话号码参数字符串
        Uri uri = Uri.parse(data);                  //将字符串转化为 Uri 实例
        Intent intent = new Intent();               //实例化 Intent
        intent.setAction(Intent.ACTION_CALL);       //设置 Intent 的 Action 属性
        intent.setData(uri);                        //设置 Intent 的 Data 属性
        startActivity(intent);                      //启动 Activity
    }

    /**
     * 启动一个拨号器的方法
     */
    public void dial() {
        String data = "tel:" + editText.getText(); //电话号码参数字符串
        Uri uri = Uri.parse(data);                     //将字符串转化为 Uri 实例
        Intent intent = new Intent();                  //实例化 Intent
```

```
        intent.setAction(Intent.ACTION_DIAL);  //设置 Intent 的 Action 属性
        intent.setData(uri);                   //设置 Intent 的 Data 属性
        startActivity(intent);                 //启动 Activity
    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.call" android:versionCode="1" android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/
    app_name">
        <activity android:name=".CallActivity" android:label="@string/
        app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="1" />
    <uses-permission android:name="android.permission.CALL_PHONE"></uses-
    permission >
</manifest>
```

5.4　使用短信消息

在 Android 手机操作系统平台中，各种各样的 Android 应用开发越来越多。其中，电话与短信服务是使用最频繁也是最多的，这一节我们介绍在 Android 中如何使用短信服务。

5.4.1　获得发送和接收短信消息的许可权

在手机应用中，短信服务是不可缺少的重要应用之一，几乎也是手机使用频率最高的应用之一。Android 手机平台中提供了发短信的类 SmsManager，通过操作这个类，就可以完成手机的短信发送与接收工作。要实现发送短信和接收短信，还需要在程序的 AndroidManifest.xml 文件中添加如下权限代码：

```
<uses-permission android:name="android.permission.SEND_SMS"></uses-perm-
ission >
<uses-permission android:name="android.permission.READ_SMS "></uses-perm-
ission >
<uses-permission android:name="android.permission.RECEIVE_SMS "></uses-
permission >
```

代码说明如下。

- android.permission.SEND_SMS：允许程序发送 SMS 短信。
- android.permission.READ_SMS：允许程序读取短信息。

- android.permission.RECEIVE_SMS：允许程序监控一个将收到短信息，记录或处理。

5.4.2　发送短信消息

本节讲解如何发送短信消息，通过 SmsManager 对象的 sentTextMessage()方法来实现完成。sentTextMessage()方法需要传入 5 个参数，依次是收件人地址（String）、发送人地址（String）、正文内容（String）、发送服务（PendingIntent）、送达服务（PendingIntent）。其中，收件人地址与正文内容是不能为 Null 的两个参数。

在这个示例中，我们用了一个按钮和两个文本输入框，它们的功能分别是发送信息、输入收件人信息、输入短信内容信息。当我们输入完成后，单击发送信息按钮，程序将发送短信，并提示短信状态。我们来看一下运行后的效果，如图 5.19 和图 5.20 所示。



图 5.19　运行效果图 1



图 5.20　运行效果图 2

实现以上效果的代码如下：

SendActivity.java

```
package com.sendsms;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class SendActivity extends Activity {
    private Button sendButton = null;  //创建发送按钮 Button 组件对象
    private EditText addressee = null; //创建收件人编辑框 EditText 组件对象
    private EditText message = null;   //创建信息内容编辑框 EditText 组件对象
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        sendButton = (Button) findViewById(R.id.send);
                                       //实例化发送按钮 Button 组件对象
        addressee = (EditText) findViewById(R.id.addressee);
                                       //实例化收件人编辑框 EditText 组件对象
        message = (EditText) findViewById(R.id.message);
                                       //实例化收件人编辑框 EditText 组件对象
        addressee.setText("请输入接收人的电话号码");   //设置默认收件人提示信息
        message.setText("请输入短信内容");         //设置默认信息内容提示信息
        //添加收件人编辑框单击事件监听
```

```
        addressee.setOnClickListener(new EditText.OnClickListener() {

            @Override
            public void onClick(View arg0) {
                addressee.setText("");
            }
        });
        //添加信息内容编辑框单击事件监听
        message.setOnClickListener(new EditText.OnClickListener() {
            @Override
            public void onClick(View arg0) {
                message.setText("");
            }
        });
        //添加发送按钮单击事件监听
        sendButton.setOnClickListener(new Button.OnClickListener() {
            @Override
            public void onClick(View arg0) {
                String strAddressee = addressee.getText().toString();
                                                    //获取收件人信息
                String strMessage = message.getText().toString();
                                                    //获取发送内容消息
                if ("".equals(strAddressee)) {      //判断收件人信息是否为空
                    showMessage("收件人信息不能为空");//调用信息提示方法
                    return;
                }
                if ("".equals(strMessage)) {        //判断发送内容是否为空
                    showMessage("信息内容不能为空");  //调用信息提示方法
                    return;
                }
                // 构建一个 Default 的 SmsManager 对象
                SmsManager smsManager = SmsManager.getDefault();
                // 构建 PendingIntent 对象，并使用 getBroadcast()方法广播
                PendingIntent pendingIntent = PendingIntent.getBroadcast(
                        SendActivity.this, 0, new Intent(), 0);
                smsManager.sendTextMessage(strAddressee, null, stressage,
                        pendingIntent, null);       //发送短信消息
                Toast.makeText(SendActivity.this, "短信发送成功", 1000).show();
                                                    //信息提示方法
            }
        });
    }

    /**
     * 提示消息弹出方法
     */
    public void showMessage(String message) {
        AlertDialog alertDialog = new AlertDialog.Builder(this).create();
                                                //创建 AlertDialog 对象
        alertDialog.setTitle("提示信息");        //设置信息标题
        alertDialog.setMessage(message);        //设置信息内容
        // 设置确定按钮，并添加按钮监听事件
        alertDialog.setButton("确定",
                new android.content.DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface arg0, int arg1) {
                    }
                });
```

```
        alertDialog.show();                    //设置弹出提示框
    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.sendsms"
     android:versionCode="1"
     android:versionName="1.0">
   <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".SendActivity"
               android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
   <uses-sdk android:minSdkVersion="1" />
   <uses-permission
android:name="android.permission.SEND_SMS"></uses-permission >
</manifest>
```

5.4.3　接收短信消息

本节讲解如何接收短信消息。通过 SmsManager 对象的 SmsManager.getDisplayOriginatingAddress()方法来获取发件人信息，通过 SmsManager.getDisplayMessageBody()方法来获取短信内容。

在这个示例中，我们用了一个空白的 Activity，只用 Toast 来弹出收到的短信信息，我们用 5.4.2 节讲述的发送短信消息，然后来接收信息。我们来看一下运行后的效果，如图 5.21 所示。



图 5.21　运行效果图

实现以上效果的代码如下：

Receiver.java

```
package com.receivedsms;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class Receiver extends BroadcastReceiver {
    private static final String SMS_RECEIVED = "android.provider.Teleph-
    ony.SMS_RECEIVED";
    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(SMS_RECEIVED)) {
                                    //判断是否是 SMS_RECEIVED 事件被触发
            StringBuilder sb = new StringBuilder();//初始化 StringBuilder
            Bundle bundle = intent.getExtras();//获取 Bundle 对象
            if (bundle != null) {               //判断 Bundle 对象是否为空
                Object[] pdus = (Object[]) bundle.get("pdus");
                SmsMessage[] msg = new SmsMessage[pdus.length];
                for (int i = 0; i < pdus.length; i++) {
                    msg[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
                }
                for (SmsMessage currMsg : msg) {
                    sb.append("发件人:");
                    sb.append(currMsg.getDisplayOriginatingAddress());
                    sb.append("\n 内容：");
                    sb.append(currMsg.getDisplayMessageBody());
                }
                Toast toast = Toast.makeText(context, "收到了短消息：\n"
                        + sb.toString(), Toast.LENGTH_LONG);
                toast.show();
            }
        }
    }
}
```

ReceivedSMS.java

```
package com.receivedsms;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ReceivedSMS extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.receivedsms" android:versionCode="1" android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/
    app_name">
        <activity android:name=".ReceivedSMS" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name=".Receiver" android:enabled="true">
```

```
            <intent-filter>
                <action android:name="android.provider.Telephony.SMS_REC-
                EIVED" />
            </intent-filter>
        </receiver>
    </application>
    <uses-sdk android:minSdkVersion="1" />
    <uses-permission android:name="android.permission.SEND_SMS"></uses-
    permission>
    <uses-permission android:name="android.permission.RECEIVE_SMS"></
    uses-permission>
</manifest>
```

5.5　使用蓝牙

在 5.4 节中学习了如何使用短信消息，本节将学习 Android 平台的蓝牙功能，了解蓝牙服务，控制本地蓝牙。

5.5.1　蓝牙服务介绍

现在手机自带蓝牙功能已经很普遍了，蓝牙是一种设备短距离无线通信技术。使用蓝牙可以搜索并连接到附近的蓝牙设备，可以在两个已经进行过配对的蓝牙设备之间进行数据的传输。

在 Android 中要管理本地蓝牙，首先要接触的是蓝牙适配器（BluetoothAdapter），BluetoothAdapter 提供了 disable()方法关闭蓝牙的方法，enable()方法打开蓝牙方法，getAddress()方法获取本地蓝牙地址，getName()方法获取本地蓝牙名称，isEnabled()方法判断蓝牙是否打开等一系列方法。获得蓝牙适配器的代码如下：

```
BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
```

在 Android 中要想使用蓝牙服务，还需要在程序的 AndroidManifest.xml 文件中添加如下权限代码：

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
```

5.5.2　控制本地蓝牙设备

这一节来学习如何控制蓝牙适配器打开或关闭蓝牙。BluetoothAdapter.disable()是关闭蓝牙的方法，BluetoothAdapter .enable()是打开蓝牙方法。但 enable()方法打开蓝牙不会弹出用户提示，更多的时候我们需要询问用户是否打开，所以如下是具有提示询问的代码：

```
Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivity(enableIntent);
```

在这个示例中，我们用了两个按钮，它们的功能分别是打开蓝牙和关闭蓝牙功能。当单击打开蓝牙按钮时，弹出提示框，询问用户是否打开蓝牙，用户选择“是”则顺利打开

蓝牙，否则不打开蓝牙。当单击关闭蓝牙按钮时，程序判断蓝牙是否为开启状态，如果蓝牙开启则将其被关闭，否则提示用户蓝牙处于关闭状态。我们来看一下运行后的效果，如图 5.22、图 5.23、图 5.24、图 5.25 和图 5.26 所示。



图 5.22　运行效果图 1



图 5.23　运行效果图 2



图 5.24　运行效果图 3



图 5.25　运行效果图 4　　　　图 5.26　运行效果图 5

实现以上效果的代码如下：

SetBluetooth.java

```
package com.setbluetooth;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class SetBluetooth extends Activity {
    private BluetoothAdapter bluetoothAdapter = null;  //本地蓝牙适配器
    private TextView statusText = null;     //蓝牙状态显示 TextView 组件对象
    private Button openButton = null;       //打开蓝牙 Button 组件对象
    private Button closeButton = null;      //关闭蓝牙 Button 组件对象
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        statusText = (TextView) findViewById(R.id.status_text);
                                   //实例化蓝牙状态显示 TextView 组件对象
        openButton = (Button) findViewById(R.id.open_button);
                                   //实例化打开蓝牙 Button 组件对象
        closeButton = (Button) findViewById(R.id.close_button);
```

```
                                    //实例化关闭蓝牙 Button 组件对象
        openButton.setOnClickListener(new Button.OnClickListener(){
                                    //为 openButton 添加单击事件监听
            @Override
            public void onClick(View arg0) {
                openBluetooth();        //调用开启蓝牙方法
                statusText.setText("蓝牙状态：开启");     //设置蓝牙状态提示
            }
        });
        closeButton.setOnClickListener(new Button.OnClickListener(){
                                    //为 closeButton 添加单击事件监听
            @Override
            public void onClick(View arg0) {
                closeBluetooth();       //调用关闭蓝牙方法
                statusText.setText("蓝牙状态：关闭");     //设置蓝牙状态提示
            }
        });
        // 得到一个本地蓝牙适配器，getDefaultAdapter()函数用于获取本地蓝牙适配器
        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        if (bluetoothAdapter == null) {     //如果适配器为 null，则不支持蓝牙
            Toast.makeText(this, "该设备不支持蓝牙", Toast.LENGTH_LONG).show();
            finish();                       //关闭程序
            return;
        }
        if (bluetoothAdapter.isEnabled()) {     //判断蓝牙是否开启
            statusText.setText("蓝牙状态：开启");//设置蓝牙状态提示
        } else {
            statusText.setText("蓝牙状态：关闭");//设置蓝牙状态提示
        }
    }

    /**
     * 打开蓝牙方法
     */
    public void openBluetooth() {
        if (!bluetoothAdapter.isEnabled()) {    //判断蓝牙是否打开
            Intent enableIntent = new Intent(
                    BluetoothAdapter.ACTION_REQUEST_ENABLE);//打开蓝牙
            startActivity(enableIntent);
            //提示蓝牙正在开启中
            Toast.makeText(this, "蓝牙开启中......", Toast.LENGTH_SHORT).show();
        } else {
            //提示蓝牙已经开启
            Toast.makeText(this, "蓝牙已经开启", Toast.LENGTH_SHORT).show();
        }
    }
    /**
     * 关闭蓝牙方法
     */
    public void closeBluetooth() {
        if (bluetoothAdapter.isEnabled()) {     //判断蓝牙是否打开
            bluetoothAdapter.disable();         //关闭蓝牙
            //提示蓝牙已经关闭
            Toast.makeText(this, "蓝牙已经关闭", Toast.LENGTH_SHORT).show();
        } else {
            //提示蓝牙是关闭的
```

```
                Toast.makeText(this, "蓝牙是关闭的", Toast.LENGTH_SHORT).show();
            }
        }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.setbluetooth" android:versionCode="1" android:versionN-
    ame="1.0">
    <application android:icon="@drawable/icon" android:label="@string/ap-
    p_name">
        <activity android:name=".SetBluetooth" android:label="@string/ap-
        p_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="1" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.BLUETOOTH" />
</manifest>
```

第 6 章　Android 本地存储系统

本章将对 Android 本地存储系统做一个详细的讲解，包括对 Android 系统文件的功能阐述、Android 系统安全敏感项的权限声明，以及 Android 程序中对文件的各种操作。想要理解并运用 Android 本地存储系统，那么必须先来了解它。

本章涉及的理论知识较多，Android 本地存储系统也是 Android 开发最常用和最关键的，特别是 Android 系统安全敏感项的权限声明。

6.1　Android 系统文件结构

“工欲善其事，必先利其器”，在学习 Android 程序中对文件的各种操作之前，让我们来了解 Android 的系统结构。Android 的系统文件，主要存储在\system 文件里，下面我们就来详细的了解\system 文件夹的结构。system 文件夹的结构如图 6.1 所示。

\system 文件夹详细说明如下。

- \system\app：\app 主要存放的是常规下载的应用程序，可以看到都是以 APK 格式结尾的 APK 文件，\app 文件夹下的程序为系统默认的组件，当然我们自己安装的软件是不会出现在这里的，而是在\data 文件夹中。
- \system\bin：\bin 文件夹下的文件都是系统的本地程序，也就是二进制的程序，主要都是 Linux 系统自带的组件（命令）。
- \system\etc：\etc 文件夹主要存储着 Android 的系统配置文件，比如 GPS 设置文件（gps.conf）、存储挂载配置文件（mountd.conf）等。
- \system\fonts：\fonts 文件夹主要是存储 Android 系统字体相关的文件，比如字体样式、中文字库、unicode 字库等。
- \system\framework：\framework 文件夹主要是存储 Android 系统核心文件、系统平台框架核心文件，比如核心库（core.jar）、系统服务（svc.jar）等。
- \system\lib：\lib 文件夹主要是存储 Android 系统底层库，比如系统服务组件（libandroid_servers.so）、蓝牙组件（libbluetooth.so）等。
- \system\media：\media 文件夹主要是存储 Android 系统提示事件音和一些常规的铃声，比如闹铃音（alarms）、提示音（notifications）等。
- \system\xbin：\xbin 文件夹主要是存储 Android 系统管理工具和配置工具。

```
system
  app
  bin
  build.prop
  etc
  fonts
  framework
  lib
  lost+found
  media
  tts
  usr
  xbin
```

图 6.1　\system 文件夹结构图

- \system\build.prop：\build.prop 文件是一个属性文件，记录 Android 系统内核、机型、版本，以及系统的设置和改变等信息。
- \system\usr：\usr 文件夹是 Android 用户文件夹，比如共享、键盘布局、时间区域文件等。
- \system\modules：\modules 文件夹主要存储是 Android 系统内核模块（主要是 fs 和 net）和模块配置文件。
- \system\lost+found：\lost+found 文件夹是基于 YAFFS（Yet Another Flash Filing System）文件系统固有的，类似回收站的文件夹。
- \system\sd：\sd 文件夹是 SD 卡中的 EXT2 分区的挂载目录。

6.2　文件访问权限

Android 程序执行时，需要读取到系统安全敏感项必需在 androidmanifest.xml 中声明相关权限请求，也就是 Android 的访问权限。在开发的过程中，我们不必把所有的权限都在 androidmanifest.xml 中声明，我们只需选取所需权限声明即可，完整的权限列表如下。

- android.permission.WRITE_APN_SETTINGS：允许程序写入 API 设置。
- android.permission.WRITE_CALENDAR：允许一个程序写入但不读取用户日历数据。
- android.permission.WRITE_CONTACTS：允许程序写入但不读取用户联系人数据。
- android.permission.WRITE_GSERVICES：允许程序修改 Google 服务地图。
- android.permission.WRITE_OWNER_DATA：允许一个程序写入但不读取所有者数据。
- android.permission.WRITE_SETTINGS：允许程序读取或写入系统设置。
- android.permission.WRITE_SMS：允许程序写短信。
- android.permission.WRITE_SYNC_SETTINGS：允许程序写入同步设置。
- android.permission.ACCESS_CHECKIN_PROPERTIES：允许读写访问"properties"表在 checkin 数据库中，该值可以修改上传。
- android.permission.ACCESS_COARSE_LOCATION：允许一个程序访问 CellID 或 Wi-Fi 热点来获取粗略的位置。
- android.permission.ACCESS_FINE_LOCATION：允许一个程序访问精确位置（如 GPS）。
- android.permission.ACCESS_LOCATION_EXTRA_COMMANDS：允许应用程序访问额外的位置提供命令。
- android.permission.ACCESS_MOCK_LOCATION：允许程序创建模拟位置。
- android.permission.ACCESS_NETWORK_STATE：允许程序访问有关 GSM 网络信息。
- android.permission.ACCESS_SURFACE_FLINGER：允许程序使用 SurfaceFlinger 底层特性。
- android.permission.ACCESS_WIFI_STATE：允许程序访问 Wi-Fi 网络状态信息。

- android.permission.ADD_SYSTEM_SERVICE：允许程序发布系统级服务。
- android.permission.BATTERY_STATS：允许程序更新手机电池统计信息。
- android.permission.BLUETOOTH：允许程序连接到已配对的蓝牙设备。
- android.permission.BLUETOOTH_ADMIN：允许程序发现和配对蓝牙设备。
- android.permission.BRICK：请求能够禁用设备（非常危险）。
- android.permission.BROADCAST_PACKAGE_REMOVED：允许程序在一个应用程序包已经移除后广播一个提示消息。
- android.permission.BROADCAST_STICKY：允许一个程序广播常用 intents。
- android.permission.CALL_PHONE：允许一个程序初始化一个电话拨号，不通过拨号用户界面要用户确认。
- android.permission.CALL_PRIVILEGED：允许一个程序拨打任何号码，包含紧急号码，无需通过拨号用户界面要用户确认。
- android.permission.CAMERA：请求访问使用照相设备。
- android.permission.CHANGE_COMPONENT_ENABLED_STATE：允许一个程序是否改变一个组件或其他的启用和禁用。
- android.permission.CHANGE_CONFIGURATION：允许一个程序修改当前设置，如本地化。
- android.permission.CHANGE_NETWORK_STATE：允许程序改变网络连接状态。
- android.permission.CHANGE_WIFI_STATE：允许程序改变 Wi-Fi 连接状态。
- android.permission.CLEAR_APP_CACHE：允许一个程序从所有安装的程序中清除缓存。
- android.permission.CLEAR_APP_USER_DATA：允许一个程序清除用户设置。
- android.permission.CONTROL_LOCATION_UPDATES：允许禁止无线模块中的位置更新提示块。
- android.permission.DELETE_CACHE_FILES：允许程序删除缓存文件。
- android.permission.DELETE_PACKAGES：允许一个程序删除包。
- android.permission.DEVICE_POWER：允许访问底层电源管理。
- android.permission.DIAGNOSTIC：允许程序 RW 诊断资源。
- android.permission.DISABLE_KEYGUARD：允许程序禁用键盘锁。
- android.permission.DUMP：允许程序从系统服务的返回状态下抓取信息。
- android.permission.EXPAND_STATUS_BAR：允许一个程序扩展收缩状态栏，Android 开发网提示应该是一个类似 Windows Mobile 中的托盘程序。
- android.permission.FACTORY_TEST：作为一个工厂测试程序，运行在 root 用户。
- android.permission.FLASHLIGHT：访问闪光灯。
- android.permission.FORCE_BACK：允许程序强行后退操作，无论是否是在顶层的 Activities。
- android.permission.FOTA_UPDATE：Android 预留权限。
- android.permission.GET_ACCOUNTS：访问一个在 AccountsService 中的账户列表。
- android.permission.GET_PACKAGE_SIZE：允许一个程序获取任何 package 占用空间容量。

- android.permission.GET_TASKS：允许一个程序获取信息，如有关当前或最近运行的任务、一个缩略的任务状态，是否活动等。
- android.permission.HARDWARE_TEST：允许访问硬件。
- android.permission.INJECT_EVENTS：允许一个程序截获用户事件（如按键、触摸、轨迹球等）到一个时间流。
- android.permission.INSTALL_PACKAGES：允许一个程序安装 packages。
- android.permission.INTERNAL_SYSTEM_WINDOW：允许打开窗口使用系统用户界面。
- android.permission.INTERNET：允许程序打开网络套接字。
- android.permission.MANAGE_APP_TOKENS：允许程序管理（创建、催后、z -order 默认向 z 轴推移）程序引用在窗口管理器中。
- android.permission.MASTER_CLEAR：允许清除一切数据。
- android.permission.MODIFY_AUDIO_SETTINGS：允许程序修改全局音频设置。
- android.permission.MODIFY_PHONE_STATE：允许修改话机状态，如电源、人机接口等。
- android.permission.MOUNT_UNMOUNT_FILESYSTEMS：允许挂载和反挂载文件系统可移动存储。
- android.permission.PERSISTENT_ACTIVITY：允许一个程序设置它的 Activities 显示。
- android.permission.PROCESS_OUTGOING_CALLS：允许程序监视、修改有关拨出电话。
- android.permission.READ_CALENDAR：允许程序读取用户日历数据。
- android.permission.READ_CONTACTS：允许程序读取用户联系人数据。
- android.permission.READ_FRAME_BUFFER：允许程序屏幕波或和更多常规的访问帧缓冲数据。
- android.permission.READ_INPUT_STATE：允许程序返回当前按键状态。
- android.permission.READ_LOGS：允许程序读取底层系统日志文件。
- android.permission.READ_OWNER_DATA：允许程序读取所有者数据。
- android.permission.READ_SMS：允许程序读取短信息。
- android.permission.READ_SYNC_SETTINGS：允许程序读取同步设置。
- android.permission.READ_SYNC_STATS：允许程序读取同步状态。
- android.permission.REBOOT：请求能够重新启动设备。
- android.permission.RECEIVE_BOOT_COMPLETED：允许程序在系统完成启动后接收到（ACTION_BOOT_COMPLETED）广播。
- android.permission.RECEIVE_MMS：允许一个程序监控将收到 MMS 彩信，记录或处理。
- android.permission.RECEIVE_SMS：允许程序监控一个将收到短信息，记录或处理。
- android.permission.RECEIVE_WAP_PUSH：允许程序监控将收到 WAP PUSH 信息。
- android.permission.RECORD_AUDIO：允许程序录制音频。
- android.permission.REORDER_TASKS：允许程序改变 Z 轴排列任务。

- android.permission.RESTART_PACKAGES：允许程序重新启动其他程序。
- android.permission.SEND_SMS：允许程序发送 SMS 短信。
- android.permission.SET_ACTIVITY_WATCHER：允许程序监控或控制系统中已经启动的全局 Activities。
- android.permission.SET_ALWAYS_FINISH：允许程序控制活动是否间接在后台完成。
- android.permission.SET_ANIMATION_SCALE：修改全局信息比例。
- android.permission.SET_DEBUG_APP：配置一个程序用于调试。
- android.permission.SET_ORIENTATION：允许底层访问设置屏幕方向和实际旋转。
- android.permission.SET_PREFERRED_APPLICATIONS：允许一个程序修改列表参数 PackageManager.addPackageToPreferred() 方法和 PackageManager.removePackageFromPreferred()方法。
- android.permission.SET_PROCESS_FOREGROUND：允许程序把当前运行程序强行运行到前台。
- android.permission.SET_PROCESS_LIMIT：允许设置最大的运行进程数量。
- android.permission.SET_TIME_ZONE：允许程序设置时间区域。
- android.permission.SET_WALLPAPER：允许程序设置壁纸。
- android.permission.SET_WALLPAPER_HINTS：允许程序设置壁纸 hits。
- android.permission.SIGNAL_PERSISTENT_PROCESSES：允许程序请求发送信号到所有显示的进程中。
- android.permission.STATUS_BAR：允许程序打开、关闭或禁用状态栏及图标。
- android.permission.SUBSCRIBED_FEEDS_READ：允许一个程序访问订阅 RSS Feed 内容提供。
- android.permission.SYSTEM_ALERT_WINDOW：允许一个程序打开窗口使用 TYPE_SYSTEM_ALERT，显示在其他所有程序的顶层。
- android.permission.VIBRATE：允许访问振动设备。
- android.permission.WAKE_LOCK：允许使用 PowerManager 的 WakeLocks 保持进程在休眠时从屏幕消失。

我们在程序实际开发过程中，需要谨慎的选择权限的声明。如我们要声明一个网络权限，能让我们的程序联网，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com." android:versionCode="1" android:versionName="1.0">
    <uses-permission android:name="android.permission.INTERNET">
    </uses-permission>
</manifest>
```

代码说明如下。

- < uses-permission>：Android 系统安全敏感项权限声明标签。
- android:name：< uses-permission>的标签元素，指定 Android 系统安全敏感项的名称。
- android.permission.INTERNET：允许程序打开网络套接字。

6.3　程序私有文件

我们都知道，操作系统如 Windows 平台下，我们的应用程序要访问或者修改其他应用程序的文件等资源，必需要取得特定的访问权限。当然，在 Android 平台下，我们的应用程序所有的数据都是私有的。

应用程序安装到 Android 系统后，这个应用程序的私有文件夹就会被创建，位于 Android 系统的/data/data/<应用程序包名>目录下，默认情况下其他的应用程序都无法在这个文件夹中写入数据。

Android 平台支持 Java 平台下的文件 I/O 操作，实现文件的存储与读取主要使用 FileOutputStream 和 FileInputStream 这两个类。下面我们通过一个例子来实现 Android 平台下的 I/O 操作。

在这个示例中，我们读取数据并显示到屏幕，其主要功能是在程序启动时，创建一个名为 test.txt 的文件，文件存放在应用程序私有的数据文件夹下，并向文件中写入自定义数据内容，然后读取 test.txt 文件中的数据内容，显示到手机屏幕中。先让我们来看一下运行后的效果，如图 6.2 所示。

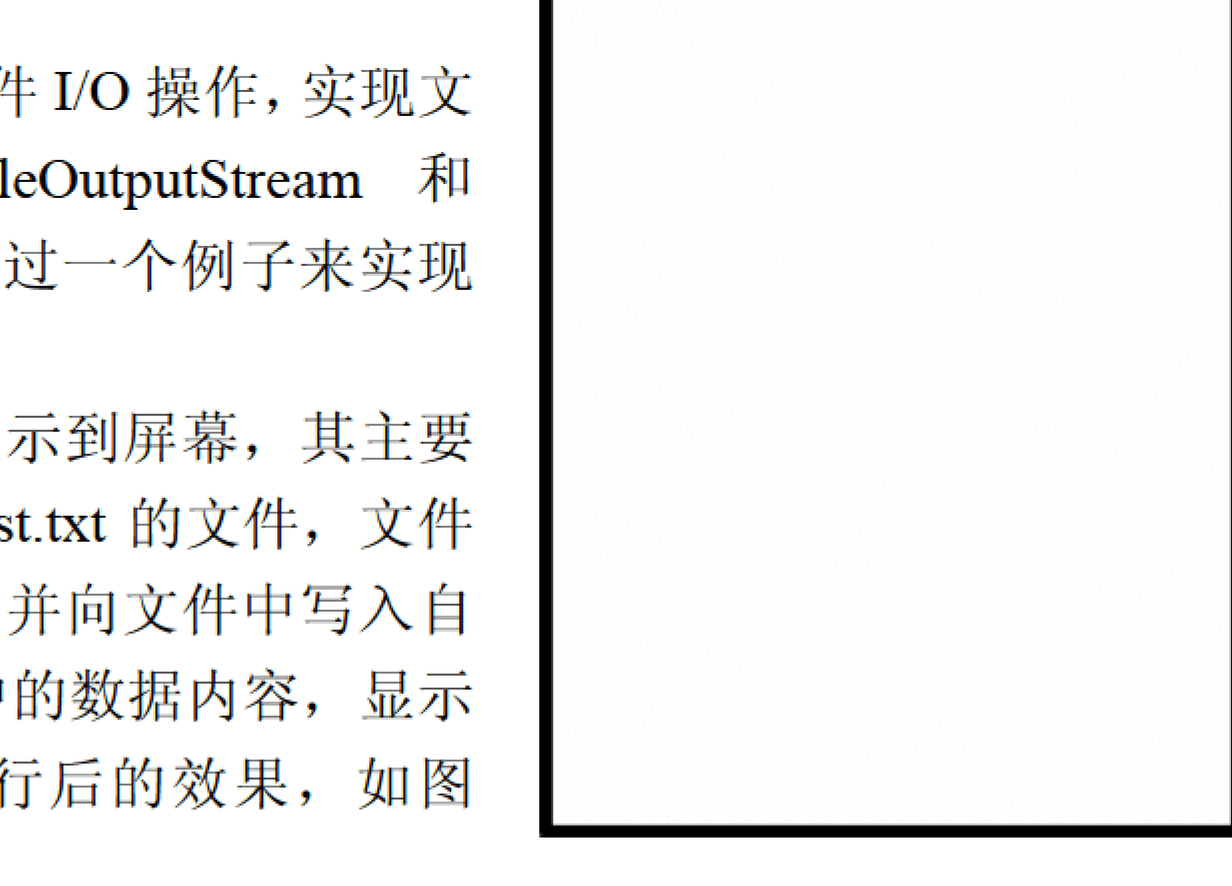


图 6.2　运行效果图

实现以上效果的代码如下：

Example.java

```
package com.example;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class Example extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.example);
        String fileName="test.txt";              //文件名称
        String content="天天向上 abcd123";       //指定数据内容
        String result="";                        //读取文件返回的 String 对象
        boolean istrue = writeFile(fileName, content);
                                                 //调用写入数据到文件的方法
        if (istrue) {                            //判断写入数据是否成功
            result+=fileName+"创建成功\n\r";
        }else {
            result+=fileName+"创建失败\n\r";
        }
        //调用读取文件方法，获取返回 String 对象
        result+=readFile(fileName);
        TextView textView = (TextView) findViewById(R.id.textView);
                                                 //初始化 TextView
        //把读取文件的返回结果显示到 TextView 中
        textView.setText(result);
```

```
    }

    /**
     * 为指定的文件中写入指定的数据
     * @param fileName 文件名称
     * @param content 指定数据内容
     * @return boolean 类型(true 表示数据写入成功，false 表示数据写入失败)
     */
    public boolean writeFile(String fileName, String content) {
        try {
            //创建 FileOutputStream 对象，MODE_PRIVATE：默认模式
            FileOutputStream fOutputStream = openFileOutput(fileName,
            MODE_PRIVATE);
            //将写入的字符串转化成 byte 数组
            byte[] buffer = content.getBytes();
            fOutputStream.write(buffer);        //将 byte 数组写入文件
            fOutputStream.flush();              //清空缓存
            fOutputStream.close();              //关闭 FileOutputStream 对象
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 读取指定文件的数据，并返回 String 对象
     * @param fileName 文件名称
     * @return String 对象
     */
    public String readFile(String fileName) {
        String result = "";                     //返回字符串结果
        try {
            FileInputStream fInputStream = openFileInput(fileName);
                                                //创建 FileInputStream 对象
            int len = fInputStream.available();     //获取文件的长度
            //创建文件长度大小的 byte 数组
            byte[] buffer = new byte[len];
            fInputStream.read(buffer);          //将文件流写入 byte 数组
            //将 byte 数组转换为 String 对象
            result = new String(buffer);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return result;        //返回字符串结果
    }
}
```

以上就是这个例子的完整代码。我们看到了把文件写入数据和读取文件信息，那么我们来看看这个程序的私有文件夹结构，如图 6.3 所示。

如图 6.3 所示，com.example 是程序的包名，在 com.example 目录下的 files 文件夹中的 test.txt 就是我们写入数据并创建的私有文件。

```
data
  anr
  app
  app-private
  backup
  dalvik-cache
  data
    com.example
      files
        test.txt
      lib
```

图 6.3　私有文件结构图

6.4　SharedPreferences 存储

SharedPreferences 存储是 Android 提供用来存储一些简单的配置信息的一种机制，其以键值对的方式存储，使得我们可以很方便地读取和存入。例如，登录的用户名或密码和一些默认的欢迎语等。

在这个示例中，我们实现了一个登录的效果。首先输入用户名和密码，单击登录按钮，程序将用户名和密码数据信息保存到自定义的 XML（user_info.xml）中，user_info.xml 中的用户数据以键值对方式存储，然后程序读取 user_info.xml 中的数据信息，并弹出信息框，提示用户登录成功。先让我们来看一下运行后的效果，如图 6.4 和图 6.5 所示。





图 6.4　程序 Shared 运行结果图 1



图 6.5　程序 Shared 运行结果图 2

SharedData.java 代码如下：

```
package com.shared;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class SharedData extends Activity {
    private String info = "user_info";          //共享文件名
    private String user = "";                   //用户名
    private String password = "";               //密码
    private EditText userText = null;           //用户名 EditText 组件对象
    private EditText passwordText = null;       //密码 EditText 组件对象

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        userText = (EditText) findViewById(R.id.user);
```

```
                                        //实例化用户名 EditText 组件对象
        passwordText = (EditText) findViewById(R.id.password);
                                        //实例化密码 EditText 组件对象
        getData();                      //调用获取文件中的数据
        Button button = (Button) findViewById(R.id.submit);
                                        //实例化登录 Button 组件对象
        //为登录 Button 组件对象添加单击事件监听
        button.setOnClickListener(new Button.OnClickListener() {
            @Override
            public void onClick(View arg0) {
                user = userText.getText().toString().trim();
                                        //获取用户输入框的值
                password = passwordText.getText().toString().trim();
                                        //获取密码输入框的值
                saveData();             //调用保存数据到文件
            }
        });
    }

    /**
     * 把 user(用户名),password(密码)保存到文件
     */
    public void saveData() {
        SharedPreferences sPreferences = getSharedPreferences(info, 0);
                                              //获取 SharedPreferences
        //打开 SharedPreferences 的编辑状态
        Editor editor = sPreferences.edit();
        editor.putString("User", user);          //存储用户名
        editor.putString("Password", password);  //存储密码
        editor.commit();                          //保存数据
        //提示用户登录成功，并获取文件中的用户信息
        new AlertDialog.Builder(SharedData.this).setTitle("登录信息")
        .setMessage(
                "用户 " + sPreferences.getString("User", "") + " 登录成功")
                .setPositiveButton("确定", new OnClickListener() {
                    @Override
                    public void onClick(DialogInterface arg0, int arg1) {
                    }
                }).show();
    }

    /**
     * 获取文件中的数据，如果文件中存在相应的数据，把该数据赋值到相应的 EditText 组件
     对象
     */
    public void getData() {
        SharedPreferences sPreferences = getSharedPreferences(info, 0);
                                        //获取 SharedPreferences
        //获取 info 文件中 User 对应的数据
        user = sPreferences.getString("User", "");
        //获取 info 文件中 Password 对应的数据
        password = sPreferences.getString("Password", "");
        //把 user 赋值给用户 EditText 组件对象
        userText.setText(user);
        //把 password 赋值给密码 EditText 组件对象
        passwordText.setText(password);
```

```
    }
}
```

以上就是这个例子的完整代码，程序主要实现了SharedPreferences 存储操作。在界面布局文件（main.xml）中，两个 TextView 分别用于显示用户名和密码；两个 EditText 分别用于输入用户名和密码；Button 登录按钮，在这里是模拟登录，单击登录，实际上是把用户信息保存到文件。那么现在我们来看看保存用户信息的文件，是否存储成功，如图 6.6 所示。

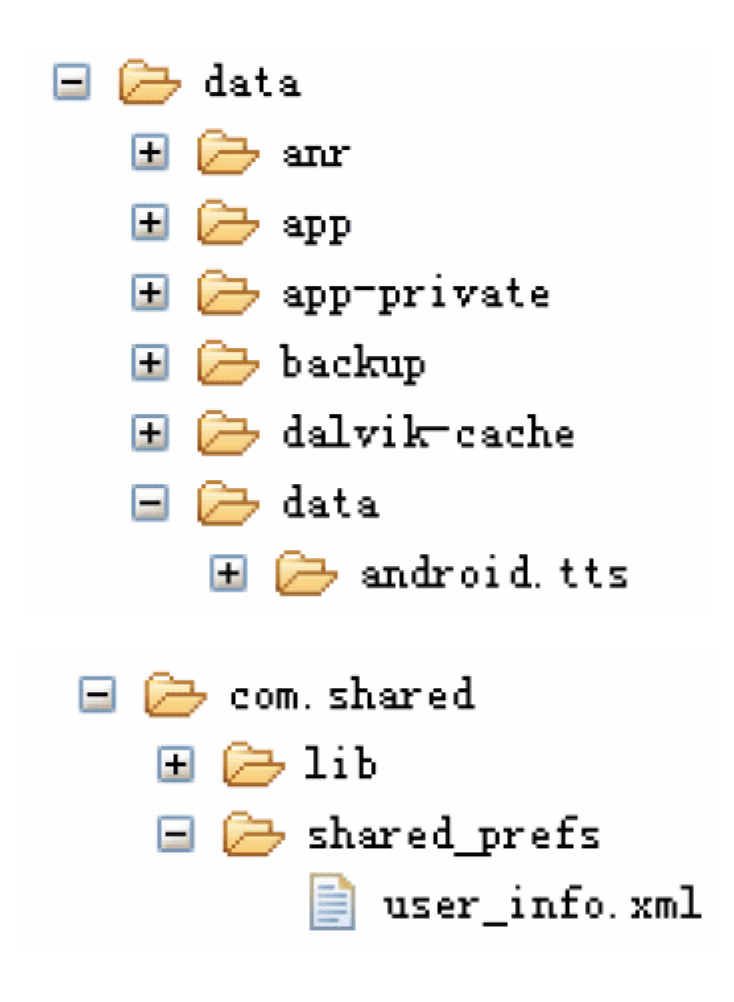


图 6.6　文件结构图

如图 6.6 所示，com.shared 是我们程序的包名，在程序包名目录下，有一个 shared_prefs 的文件夹，该文件夹存储着 XML 格式的数据文件。user_info.xml 就是程序存储用户信息的文件。

我们知道 SharedPreferences 是以键值对来存储应用程序的配置信息的一种方式，它只能存储基本数据类型，那我们来具体看看 user_info.xml 文件。

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="User">admin</string>
<string name="Password">123</string>
</map>
```

代码说明如下。

- <string name="User">：这里定义了一个字符串，名称是 User，值是 admin。这就是我们先登录输入的用户信息。
- <string name="Password">：这里是密码字符串的定义，值是 123。

我们已经看到 user_info.xml 文件中的具体内容，和我们输入的信息相吻合，那么操作成功。再次运行程序，我们上次登录所用的用户名和密码就自动显示在输入框中。

6.5　遍历文件夹

在 Android 中的 I/O 操作与 Java 中实现 I/O 操作大体相同，这一节讲解 Android 中怎么遍历文件夹。下面我们做一个例子来实现一个类似文件管理器的 Android 应用程序。

在这个示例中，我们实现了一个文件浏览器的效果。当程序启动时，默认列出当前 SDCard 中的文件目录列表，单击其中的文件夹进入该文件夹中的目录列表，单击手机返回按钮，回到上级目录。在示例中，我们要用到 File 类、Environment 类，File 类用于操作文件和目录，Environment 类提供访问环境变量。先让我们来看一下运行后的效果，如图 6.7 所示。

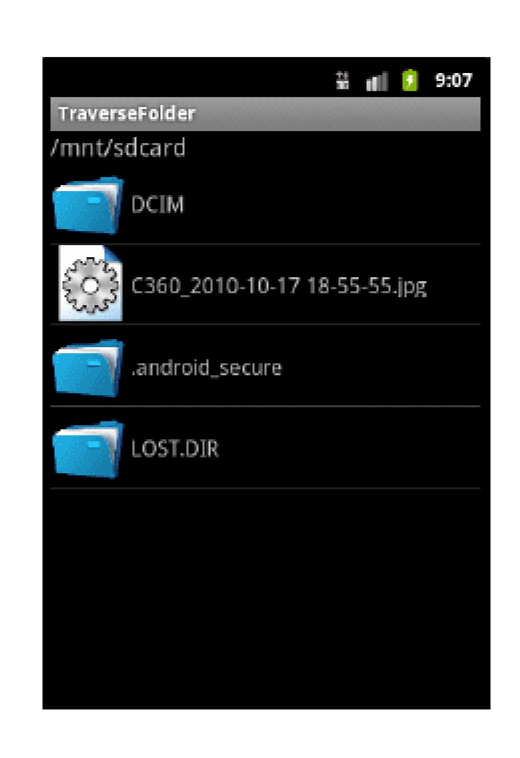


图 6.7　运行效果图

TraverseFolder.java 代码如下：

```
package com.folder;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class TraverseFolder extends Activity {
    private TextView textView = null; //用于显示目录结构的 TextView 组件对象
    private File[] files = null;            //File 数组
    private ListView listView = null;       //用于显示文件的 ListView 组件对象
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        listView = (ListView) findViewById(R.id.listView);
                                            //实例化 ListView 组件对象
        textView = (TextView) findViewById(R.id.text_view);
                                            //实例化 TextView 组件对象
        boolean sdStatus = getStorageState();//调用获取手机 SDCard 的存储状态
        //判断 SDCard 的存储状态，如果是 false，提示并结束本程序
        if (!sdStatus) {
            AlertDialog alertDialog = new AlertDialog.Builder(
                    TraverseFolder.this).create(); //创建 AlertDialog 对象
            alertDialog.setTitle("提示信息");       //设置信息标题
            alertDialog.setMessage("未安装 SD 卡，请检查你的设备");//设置信息内容
            //设置确定按钮，并添加按钮监听事件
            alertDialog.setButton("确定", new OnClickListener() {

                @Override
                public void onClick(DialogInterface arg0, int arg1) {
                    TraverseFolder.this.finish();  //结束应用程序
                }
            });
            alertDialog.show();                     //设置弹出提示框
        }

        Intent intent = getIntent();                //获取 Intent
        //获取 CharSequence 对象
        CharSequence charSequence = intent.getCharSequenceExtra("filePath");
        //判断 CharSequence 对象是否为空，为空就获取 SDCard 根目录，否则就获取传过来
          的文件目录
        if (charSequence != null) {
            File file = new File(charSequence.toString()); //实例化 File
            textView.setText(file.getPath());//更新 TextView 组件显示的目录结构
            files = file.listFiles();        //获取该目录的所有文件及目录
        } else {
            //获取 SDCard 根目录 File 对象
            File sdCardFile = Environment.getExternalStorageDirectory();
            textView.setText(sdCardFile.getPath());
                                            //设置 TextView 组件显示的目录结构
            files = sdCardFile.listFiles(); //获取 SDCard 根目录的所有文件及目录
        }

        List<HashMap<String, Object>> list = getList(files);
                                            //调用获取相应的集合
        //调用构造适配器并为 ListView 添加适配器
        setAdapter(list, files);
        listView.setOnItemClickListener(new OnItemClickListener() {
                                            //为 ListView 添加单击监听
```

```
            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                    long arg3) {
                if (files[arg2].isDirectory()) {//判断所单击的文件是否是文件夹
                    //获取该单击文件夹下的所有文件及文件夹
                    File[] childFiles = files[arg2].listFiles();
                    if (childFiles != null && childFiles.length >= 0) {
                                                //判断该单击文件夹数组不为空
                        Intent intent = new Intent();   //初始化 Intent
                        intent.setClass(TraverseFolder.this,
                                TraverseFolder.class);
                                                //指定 intent 对象启动的类
                       intent.putExtra("filePath", files[arg2]. getPath());
                                                //函数传递
                        startActivity(intent);          //启动新的 Activity
                    }
                }
            }
        });
    }

    /**
     * 构造适配器并为 ListView 添加适配器
     *
     * @param list
     *            HashMap 的集合
     * @param files
     *            File 数组
     */
    public void setAdapter(List<HashMap<String, Object>> list, File[] files) {
        SimpleAdapter simpleAdapter = new SimpleAdapter(TraverseFolder.
        this,list, R.layout.folder_list, new String[] { "image_view",
                        "folder_name" }, new int[] { R.id.image_view,
                        R.id.folder_name });        //实例化 SimpleAdapter
        listView.setAdapter(simpleAdapter);         //为 ListView 添加适配器
        this.files = files;                         //把当前 File 数组赋值
    }

    /**
     * 获取手机 SDCard 的存储状态
     *
     * @return 手机 SDCard 的存储状态(true/false)
     */
    public boolean getStorageState() {
        if (Environment.getExternalStorageState().equals(
                Environment.MEDIA_MOUNTED)) {  //判断手机 SDCard 的存储状态
            return true;
        } else {
            return false;
        }
    }

    /**
     * 根据 File[]获取相应的集合
     * @param files File 数组
     * @return List<HashMap<String, Object>>集合
     */
```

```
    public List<HashMap<String, Object>> getList(File[] files) {
        //创建 List 集合
        List<HashMap<String, Object>> list = new ArrayList<HashMap<String,
        Object>>();
        for (int i = 0; i < files.length; i++) {   //循环 File 数组
            //创建 HashMap
            HashMap<String, Object> hashMap = new HashMap<String, Object>();
            if (files[i].isDirectory()) {       //判断该文件是否是文件夹
                hashMap.put("image_view", R.drawable.dir1);
                                                //往 HashMap 中添加文件夹图片
            } else {
                hashMap.put("image view", R.drawable.file2);
                                                //往 HashMap 中添加文件图片
            }
            hashMap.put("folder_name", files[i].getName());
                                                //往 HashMap 中添加文件名
            list.add(hashMap);                  //将 HashMap 添加到 List 集合
        }
        return list;                            //返回 List 集合
    }
}
```

以上是这个例子的完整代码。TraverseFolder.java 是实现这个例子的主要 Java 类，应用 AlertDialog 来进行消息提示，AlertDialog 对象没有构造函数，所以我们不能用 new AlertDialog(…)来初始化。要想获取 AlertDialog 对象，只能用 AlertDialog. Builder(this).create()来创建 AlertDialog 对象，用 AlertDialog 的 show()方法来弹出 AlertDialog，用 AlertDialog 的 dismiss()方法来取消 AlertDialog。

应用 Intent 来实现组件间的相互调用，Intent 提供 Activity 直接交互的抽象描述，可以被 startActivty 运行。Intent 提供了一个媒介，提供组件的相互调用，实现调用者和被调用者之间的解耦。

6.6　读/写文件

在 Java 中，I/O 的 API 提供了 java.io.InputStream、java.io.OutputStream 对象来读写文件。在 Android 中也是一样，下面我们做一个例子来实现读/写文件。

在这个示例中，用户将要创建的文件名输入文本框（EditText），文件内容输入文本框（EditText），单击按钮（Button），程序将自动创建该文件，创建成功，文件名将被列在列表（ListView）中；单击文件名，将展示我们创建的文件。先让我们来看一下运行后的效果，如图 6.8、图 6.9、图 6.10 和图 6.11 所示。

MainActivity.java 代码如下：

```
package com.file;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class MainActivity extends Activity {
    private ListView listView = null; //用于显示文件列表的 ListView 组件对象
    private File[] files = null;           //File 数组
    private Button createButton = null;    //创建按钮的 Button 组件对象
    private String dirPath = "";           //文件读/写指定目录
    @Override
```

图 6.8　运行效果图 1



图 6.9　运行效果图 2

图 6.10　运行效果图 3

图 6.11　运行效果图 4

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);  //为当前活动的 Activity 设置一个视图
        listView = (ListView) findViewById(R.id.listView);
                                  //实例化 ListView 组件对象
        createButton = (Button) findViewById(R.id.createButton);
                                  //实例化 Button 组件对象
        setData();                //加载数据
        listView.setOnItemClickListener(new OnItemClickListener() {
                                  //为 ListView 添加单击监听
              @Override
```

```
                public void onItemClick(AdapterView<?> arg0, View arg1,
                        int arg2, long arg3) {
                    Intent intent = new Intent();  //初始化 Intent
                    intent.setClass(MainActivity.this, ShowActivity.
                    class);        //指定 intent 对象启动的类
                    intent.putExtra("filePath", files[arg2].
                    getPath());    //函数传递
                    startActivity(intent);         //启动新的 Activity
                }
            });

        createButton.setOnClickListener(new Button.OnClickListener() {
                                        //为 Button 添加单击监听
            @Override
            public void onClick(View arg0) {
                Intent intent = new Intent();          //初始化 Intent
                intent.setClass(MainActivity.this, CreateActivity.
                class);                    //指定 intent 对象启动的类
                intent.putExtra("dirPath", dirPath);   //函数传递
                startActivity(intent);                 //启动新的 Activity
            }
        });
    }

    /**
     * 填充数据
     */
    public void setData() {
        boolean sdStatus = getStorageState();//调用获取手机 SDCard 的存储状态
        //判断 SDCard 的存储状态，如果是 false，提示并结束本程序
        if (!sdStatus) {
            AlertDialog alertDialog = new AlertDialog.Builder
            (MainActivity.this)
                    .create();                  //创建 AlertDialog 对象
            alertDialog.setTitle("提示信息");    //设置信息标题
            alertDialog.setMessage("未安装 SD 卡，请检查你的设备");//设置信息内容
            //设置确定按钮，并添加按钮监听事件
            alertDialog.setButton("确定",new OnClickListener() {
                @Override
                public void onClick(DialogInterface arg0, int arg1) {
                    MainActivity.this.finish();//结束应用程序
                }
            });
            alertDialog.show();                 //设置弹出提示框
        }

        File sdCardFile = Environment.getExternalStorageDirectory();
                                   //获取 SDCard 根目录 File 对象
        dirPath = sdCardFile.getPath() + File.separator + "FileIO";
                                   //指定文件存放目录
        File dirFile = new File(dirPath);
        if (!dirFile.exists()) {           //判断文件存放目录是否存在
            dirFile.mkdir();               //创建文件存放目录
        }
        files = dirFile.listFiles();      //获取文件存放目录中的文件 File 对象
        List<HashMap<String, Object>> list = getList(files);
                                   //调用获取相应的集合
```

```
        setAdapter(list, files);    //调用构造适配器并为 ListView 添加适配器
    }

    /**
     * 获取手机 SDCard 的存储状态
     *
     * @return 手机 SDCard 的存储状态(true/false)
     */
    public boolean getStorageState() {
        if (Environment.getExternalStorageState().equals(
                Environment.MEDIA_MOUNTED)) {   //判断手机 SDCard 的存储状态
            return true;
        } else {
            return false;
        }
    }

    /**
     * 根据 File[]获取相应的集合
     *
     * @param files
     *            File 数组
     * @return List<HashMap<String, Object>>集合
     */
    public List<HashMap<String, Object>> getList(File[] files) {
        List<HashMap<String, Object>> list = new ArrayList<HashMap<String,
        Object>>();                                    //创建 List 集合
        for (int i = 0; i < files.length; i++) {     //循环 File 数组
           HashMap<String, Object> hashMap = new HashMap<String, Object>();
                                                       //创建 HashMap
            hashMap.put("file name", files[i].getName());
                                                  //往 HashMap 中添加文件名
            list.add(hashMap);                    //将 HashMap 添加到 List 集合
        }
        return list;                              //返回 List 集合
    }

    /**
     * 构造适配器并为 ListView 添加适配器
     *
     * @param list
     *            HashMap 的集合
     * @param files
     *            File 数组
     */
    public void setAdapter(List<HashMap<String, Object>> list, File[] files) {
        SimpleAdapter simpleAdapter = new SimpleAdapter(MainActivity.this,
                list, R.layout.file_list, new String[] { "file_name" },
                new int[] { R.id.fileName });         //实例化 SimpleAdapter
        listView.setAdapter(simpleAdapter);           //为 ListView 添加适配器
    }
}
```

CreateActivity.java

```
package com.file;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
```

```
public class CreateActivity extends Activity {
    private EditText nameEditText = null;    //文件名称的 EditText 组件对象
    private EditText contentEditText = null;//文件内容的 EditText 组件对象
    private Button createFileButton = null; //创建文件按钮的 Button 组件对象
    private String dirPath = "";            //文件读/写指定目录
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.create_file);//为当前活动的 Activity 设置一个视图
        nameEditText = (EditText) findViewById(R.id.createName);
                                            //实例化 EditText 组件对象
        contentEditText = (EditText) findViewById(R.id.createContent);
                                            //实例化 EditText 组件对象
        createFileButton = (Button) findViewById(R.id.createFileButton);
                                            //实例化 Button 组件对象
        Intent intent = getIntent();        //获取 Intent
        dirPath = intent.getCharSequenceExtra("dirPath").toString();
                                            //获取文件存放目录
        createFileButton.setOnClickListener(new Button.OnClickListener(){
                                            //为 Button 添加单击监听
                    @Override
                    public void onClick(View arg0) {
                        String fileName = nameEditText.getText().
                        toString();         //获取输入文件名称
                        String fileContent = contentEditText.getText()
                                .toString();//获取输入文件内容
                        boolean isTrue = writeFile(fileName, fileContent);
                                            //调用写文件方法
                        if (isTrue) {       //判断文件是否创建成功
                            //调用消息弹出方法，弹出写文件成功消息提示
                            showMessage("创建文件成功");
                        } else {
                            //调用消息弹出方法，弹出写文件失败消息提示
                            showMessage("创建文件失败");
                        }
                    }
                });
    }

    /**
     * 写文件
     *
     * @param fileName
     *            文件名称
     * @param fileContent
     *            文件内容
     * @return
     */
    public boolean writeFile(String fileName, String fileContent) {
        try {
            File file = new File(dirPath + File.separator + fileName +
            ".txt");                        //根据文件路径，创建.txt 文本文件
            OutputStream outputStream = new FileOutputStream(file);
                                            //实例化 OutputStream 对象
            byte[] contents = fileContent.getBytes();
                                            //把字符串内容转化为 byte 数组
            outputStream.write(contents);
```

```
                //将 contents.length 个字节从指定的 contents 数组写入此输出流
            outputStream.flush();   //刷新此输出流并强制写出所有缓冲的输出字节
            outputStream.close();   //关闭此输出流并释放与此流有关的所有系统资源
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 消息提示并跳转到首页
     *
     * @param message
     *            消息内容
     */
    public void showMessage(String message) {
        //创建消息提示框 AlertDialog 对象
        //setTitle：设置消息标题
        //setMessage：设置消息内容
        //setNegativeButton：设置消息确定按钮，并定义按钮单击事件
        //show：显示消息
        new AlertDialog.Builder(CreateActivity.this).setTitle("提示信息")
                .setMessage(message).setNegativeButton("确定",
                        new OnClickListener() {
                            @Override
                            public void onClick(DialogInterface arg0, int
                            arg1) {
                                Intent intent = new Intent();//实例化 Intent
                                intent.setClass(CreateActivity.this,
                                        MainActivity.class);
                                        //指定 intent 对象启动的类
                                startActivity(intent);  //启动新的 Activity
                            }
                        }).show();
    }
}
```

ShowActivity.java

```
package com.file;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ShowActivity extends Activity {
    private TextView nameTextView = null;      //文件名称的 TextView 组件对象
    private TextView contentTextView = null;   //文件内容的 TextView 组件对象
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.show_file);//为当前活动的 Activity 设置一个视图
        nameTextView = (TextView) findViewById(R.id.showName);
                                        //实例化文件名称的 TextView 组件对象
        //实例化文件内容的 TextView 组件对象
        contentTextView = (TextView) findViewById(R.id.showContent);
        Intent intent = getIntent();      //获取 Intent
        String filePath = intent.getCharSequenceExtra("filePath").
        toString();                        //获取文件路径
        String name = new File(filePath).getName();    //获取文件名
```

```
            String content = readFile(filePath); //调用读取文件方法，获取文件内容
            nameTextView.setText("文件名称： " + name);          //设置文件名称
            contentTextView.setText("文件内容： " + content);    //设置文件内容
        }

        /**
         * 读文件
         * @param filePath 文件路径
         * @return
         */
        public String readFile(String filePath) {
            try {
                File file = new File(filePath);     //实例化 File 对象
                InputStream inputStream = new FileInputStream(file);
                                                    //实例化 InputStream 对象
                //inputStream.available()获取此输入流下一个方法调用可以不受阻塞地从
                此输入流读取（或跳过）的估计字节数
                byte[] b = new byte[inputStream.available()];
                inputStream.read(b);//从输入流中读取字节，并将其存储在缓冲区数组 b 中
                inputStream.close();//关闭此输入流并释放与该流关联的所有系统资源
                return new String(b);  //返回字符串内容
            } catch (Exception e) {
                e.printStackTrace();
                return "";
            }
        }
    }
```

以上是这个例子的完整代码。MainActivity.java 是实现程序首页的 Java 类，主要是展示文件的动态列表。应用了 ListView 来动态显示文件列表；Environment.getExternalStorageState()方法判断了手机内存卡的状态；AlertDialog 来进行消息提示；Intent 来实现组件间的相互调用。

CreateActivity.java 是实现程序创建文件页的 Java 类，主要是实现文件的创建。getIntent()获取 Intent 从而获取页面的参数，EditText 组件对象获取用户输入的文件名及文件内容，OutputStream 对象实现文件的创建，文件创建成功以 AlertDialog 来进行消息提示。

ShowActivity.java 是实现文件内容展示页的 Java 类，主要是文件内容的展示。TextView 组件对象用来展示文件名及文件内容，InputStream 对象实现对文件的读取操作。

第 7 章　Android 中的数据库

在应用程序中，经常会遇到需要存储数据的情况。通常情况下，会选择把数据存储在数据库中。在 Android 应用程序中，可以通过网络传输把数据存储到远端的数据库中。不过，Android SDK 提供了另外一种选择，可以把数据存储在 Android SDK 附带的 SQLite 数据库中。

7.1　创建 SQLite 数据库及表

SQLite 是一款以嵌入式为目的的设计的轻型数据库，运行起来占用的资源非常小，通常只需要几百 KB 就可以支持它的各项功能。SQLite 具有良好的兼容性，支持标准的 SQL 语句，通过 Android SDK 提供的接口，可以很轻松地使用它。

本节中，就来研究如何在 Android 应用系统中使用 SQLite 数据库。和使用其他数据库一样，首先要关注的是如何创建一个新的数据库，以及在数据库中创建新的表。

这一节的重点不在界面控件上，所以只需创建一个极其简单的界面。界面上只有两个按钮，分别用来创建数据库和表。

在 Android 应用程序中创建 SQLite 数据库，是一件非常简单的事情，简单到只需要一条语句，代码如下：

```
public class MySQLite extends Activity{
    private Button baseButton;
    private Button tableButton;
    private final String dbName = "mydb";
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.main);
        baseButton = (Button) findViewById(R.id.base); //实例化 Button 对象
        baseButton.setOnClickListener(new OnClickListener() {
                                    //为 Button 对象添加监听
            public void onClick(View v) {
                openOrCreateDatabase(dbName,MODE_PRIVATE,null);
                                    //创建名为“mydb”的数据库
            }
        });

        tableButton = (Button) findViewById(R.id.table);
                                    //实例化 Button 对象
        tableButton.setOnClickListener(new OnClickListener() {
                                    //为 Button 对象添加监听
            public void onClick(View v) {

            }
```

```
            });
        }
}
```

代码说明：

- openOrCreateDatabase()是 android.content.ContextWrapper 类的方法，而 Activity 是 ContextWrapper 的子类，MySQLite 类继承了 Activity 类，所以可以直接使用该方法。
- 从名字就可以看出，openOrCreateDatabase()是打开或创建数据库。它的作用是打开指定的数据库，如果该数据库不存在，则创建它。
- openOrCreateDatabase 方法有 3 个参数，第一个是数据的名字，用字符串表示；第二个参数是该数据的类型，是个整数常量；第三个参数是 SQLiteDatabase. CursorFactory 类型的对象，使用 null 表示用默认的 CursorFactory 创建数据库。
- MODE_PRIVATE 表示创建的数据库只能被当前创建它的应用程序使用。该参数还有另外两个选择，一个是 MODE_WORLD_READABLE，表示创建的数据库允许所有应用程序读取数据；另一个是 MODE_WORLD_WRITEABLE，表示创建的数据库允许所有的应用程序写入数据。

当数据库被创建出来后，在手机或模拟器的 data/data/应用程序目录下就可以看到数据库文件。用一个数据库文件表示一个数据库，这一点和微软的 Access 十分类似，效果如图 7.1 所示。

Name	Size	Date	Time	Permissions	Info
▷ com.android.spare_parts		2010-12-10	08:01	drwxr-x--x	
▷ com.android.speechrecorde		2010-12-10	08:01	drwxr-x--x	
▷ com.android.systemui		2010-12-10	08:01	drwxr-x--x	
▷ com.android.term		2010-12-10	08:01	drwxr-x--x	
▷ com.android.wallpaper.livep		2010-12-10	08:01	drwxr-x--x	
◢ com.example		2011-01-15	08:49	drwxr-x--x	
◢ databases		2011-01-15	08:49	drwxrwx--x	
mydb	3072	2011-01-15	08:49	-rw-rw----	
▷ lib		2011-01-15	08:48	drwxr-xr-x	
▷ com.google.android.apps.m		2010-12-12	02:45	drwxr-x--x	
▷ com.google.android.gsf		2010-12-10	08:04	drwxr-x--x	
▷ com.google.android.locatior		2010-12-10	08:01	drwxr-x--x	
▷ com.google.android.street		2010-12-10	08:01	drwxr-x--x	

图 7.1　数据库创建位置

openOrCreateDatabase()方法会返回一个 android.database.sqlite.SQLiteDatabase 对象，通过这个数据库对象，就可以执行建表、插入数据等数据库操作了。

下面就来看看如何通过 SQLiteDatabase 对象创建表，需要对原来的程序进行大幅度的修改，代码如下：

```
public class MySQLite extends Activity{
    private Button baseButton;
    private Button tableButton;
    private final String dbName = "mydb";
    private final String tableName = "users";
    private SQLiteDatabase db=null;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.main);
        baseButton = (Button) findViewById(R.id.base); //实例化 Button 对象
        baseButton.setOnClickListener(new OnClickListener() {
```

```
                                                        //为 Button 对象添加监听
            public void onClick(View v) {
                //创建名为“mydb”的数据库，并将创建的对象赋值给 db
                db=openOrCreateDatabase(dbName,MODE_PRIVATE,null);
            }
        });

        tableButton = (Button) findViewById(R.id.table);
                                                        //实例化 Button 对象
        tableButton.setOnClickListener(new OnClickListener() {
                                                        //为 Button 对象添加监听
            public void onClick(View v) {
                if(db!=null){
                    creatTable();                       //开始创建数据库表
                }
            }
        });
    }

    public void creatTable(){
        //创建表的 SQL 语句，创建一个名为 users 的表，该表有 uname 和 pwd 两个字段
        String sql = "CREATE TABLE IF NOT EXISTS " + tableName
         + " (uname VARCHAR(50), pwd VARCHAR(50));";
        db.execSQL(sql);                                //执行 sql 语句
        //查询 sqlite_master 表中类型为 table 的记录的 name 字段
        Cursor cursor = db.query("sqlite_master", new String[] { "name" },
                "type = ?", new String[] { "table" }, null, null, null, null);
        String tables="";
        if(cursor.getCount()!=0){                       //判断查询结果的条数是否为 0
            cursor.moveToFirst();                       //游标指向第一条记录
            for(int i=0;i<cursor.getCount();i+=1){
                tables=tables+cursor.getString(0)+" ";          //累加字符串
                cursor.moveToNext();                    //游标下移
            }
        }
        //把累加的结果显示在一个信息框中
        new AlertDialog.Builder(MySQLite.this).setTitle("Message")
        .setMessage(tables).setNegativeButton("确定",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog,int which) {
                }
        }).show();
    }
}
```

代码说明：

- execSQL()是 SQLiteDatabase 对象的一个方法，该方法可以执行一条标准的 SQL 语句，用来创建表，或者操作表中的数据。但是 execSQL()方法的返回值是 void，所以不能进行查询操作。
- query 是 SQLiteDatabase 对象的查询方法。SQLiteDatabase 还提供了 insert()、update()、delete()等方法来处理数据，详细的使用方法，会在后续的章节中讨论。
- SQLiteDatabase 的查询操作会返回一个 Cursor 对象，包含了查询出来的各种信息，具体使用方法会在后续的章节中讨论。
- sqlite_master 表是 SQLite 数据库中的管理表，用来定义数据库的模式。这个表是

只读的，用户不能对它执行添加、更新或删除操作。sqlite_master 表中的数据会在用户创建或删除表、索引的时候自动更新。

代码中的 creatTable()方法，首先在数据库中创建一个名为 users 的表，然后把当前数据库中的所有表的名字查询出来显示在信息框中，运行效果如图 7.2 所示。

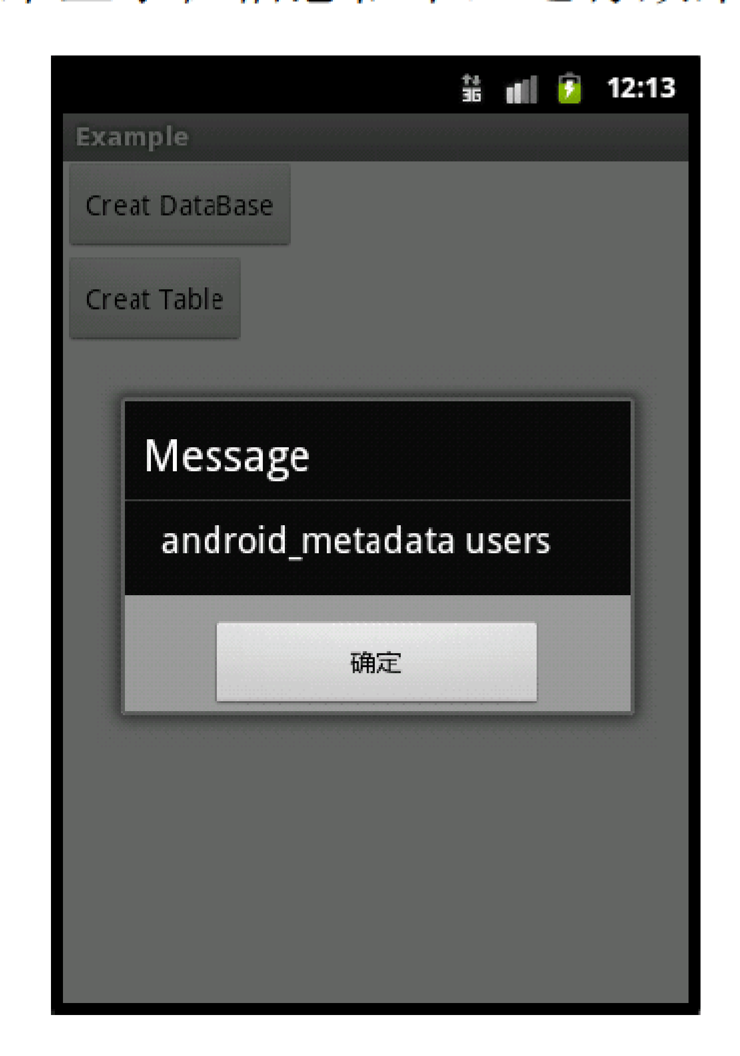


图 7.2　创建表

7.2　对表中数据的添加、删除、修改

在 Android 应用程序中，要对表中的数据进行添加、删除、修改的操作，最直接的方法就像 7.1 节中创建表一样，通过 SQLiteDatabase 对象的 execSQL()方法来执行准备好的 SQL 语句，代码如下：

```
public void executeData(){
    String sql = "insert into "+tableName+" values ('张三','123456')";
                                              //添加一条记录
    db.execSQL(sql);

    sql = "update "+tableName+" set pwd='654321' where uname='张三'";
                                              //修改一条记录
    db.execSQL(sql);

    sql = "delete from "+tableName+" where uname='张三'";  //删除一条记录
    db.execSQL(sql);
}
```

拼接可执行的 SQL 语句的方法虽然很直接，但是当表中的字段很多或者有多个限制条件的时候，这种操作会变得很繁琐，容易出错误，而且这种方法无法执行查询语句。因此 Android SDK 提供了另外一套方法，代码如下：

```
public void executeData(){
    String sql = "insert into "+tableName+" values ('张三','123456')";
                                          //添加一条记录的 sql 语句
```

```
    db.execSQL(sql);                        //执行 sql 语句
    ContentValues cv=new ContentValues();   //实例化 ContentValues 对象
    cv.put("uname", "李四");                 //插入字段值
    cv.put("pwd", "987654");                //插入字段值
    db.insert(tableName, null, cv);         //执行 insert()方法
    sql = "update "+tableName+" set pwd='654321' where uname='张三'";
                                            //修改一条记录的 sql 语句
    db.execSQL(sql);                        //执行 sql 语句
    cv=new ContentValues();                 //实例化 ContentValues 对象
    cv.put("pwd", "456789");                //插入字段值
    db.update(tableName, cv, "uname=?",new String[]{"李四"});
                                            //执行 update()方法
    sql = "delete from "+tableName+" where uname='张三'";  //删除一条记录
    db.execSQL(sql);                        //执行 sql 语句
    db.delete(tableName, "uname=?", new String[]{"李四"});//执行 delete 操作
}
```

代码说明：

- insert 方法带有 3 个参数，第 1 个是要插入数据的表名，字符串形式；第 2 个为空字段的名称，字符串形式；第 3 个是要插入的数据内容，ContentValues 对象。
- ContentValues 是一个 map 形式的集合，用来保存一条记录的字段信息。key 为字段的名称，value 为该字段的值。
- Update 方法带有 4 个参数，第 1 个是要插入数据的表名，字符串形式；第 2 个是要更新的数据内容，ContentValues 对象；第 3 个是更新条件，字符串形式；第 4 个是更新条件的值，字符串数组形式。
- 更新条件的字符串中的“？”表示一个占位符，其具体的数值由第 4 个参数提供。如果直接将参数写入到字符串中，如“uname=‘李四’”的形式，那么第 4 个参数写 null 即可。
- Delete 方法带有 3 个参数，第 1 个是要插入数据的表名，字符串形式；第 2 个是删除条件，字符串形式；第 3 个是删除条件的值，字符串数组形式。

7.3　对表中数据的查询

查询一向是对数据的各项操作中变化最多最复杂的操作，在前面的章节中，已经简单地接触了一些，本节将会具体详细地来研究。

查询操作存在的意义，在于需要获取从数据库中获得的数据，所以返回值为 void 的 execSQL()方法是不适合的。需要使用的是 query()方法，该方法会返回一个 Cursor 对象，通过对这个 Cursor 对象的各种操作，就可以获得所需的数据。

SQLiteDatabase 对象的 query()方法带有很多参数，如果只想做一个简单查询，比如查询出 users 表中的所有记录。如果是 SQL 语句的话，就是下面这种形式：

```
select * from users
```

这种情况的查询下，query()方法的参数设置很简单，代码如下：

```
public void queryData(){
```

```
    Cursor cursor = db.query(tableName, null, null,null, null, null,
    null,null);                              //执行 query 操作获得 Cursor 对象
    String str="";
    if(cursor.getCount()!=0){                //判断返回的记录条数
        cursor.moveToFirst();                //游标指向第一条记录
        for(int i=0;i<cursor.getCount();i+=1){
            str=str+cursor.getString(0)+" "+cursor.getString(1)+"\n";
                                             //获取一条记录的每个字段的值
            cursor.moveToNext();             //游标指向下一条记录
        }
    }
    //在信息框上显示所有记录信息
    new AlertDialog.Builder(MySQLite.this).setTitle("Message")
    .setMessage(str).setNegativeButton("确定",
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog,int which) {
            }
    }).show();
}
```

代码说明：

- query 方法有多种重载形式，通常情况下，使用带 8 个参数的方法。第 1 个参数是查询表的名字，字符串形式；第 2 个参数是查询的字段名，字符串数组形式；第 3 个参数是查询条件，字符串形式；第 4 个参数是查询条件的值，字符串数组形式；第 5 个参数是分组字段名，字符串形式；第 6 个参数是分组后筛选条件，字符串形式；第 7 个参数是排序字段名，字符串形式；第 8 个是查询结果返回记录条数限制，字符串形式。
- Cursor 对象包含了查询结果，并提供了多种方法来操作这些数据。
- 当 Cursor 对象处理某一条记录的时候，需要将游标指向该条记录。Cursor 对象提供了 moveToFirst()、moveToLast()、moveToNext()、moveToPrevious()方法将游标指向结果集的第一条、最后一条、下一条、上一条。也可以使用 moveToPosition()方法移动到指定位置，该方法需要一个整数位参数。
- 刚从 query()方法获得 Cursor 对象时，Cursor 对象的游标并非指向第一行记录，而是指向第一条记录的上一行。可以通过 isBeforeFirst()或 isAfterLast()方法判断游标是否指向第一条记录之前或最后一条记录之后。也可以通过 isFirst()或 isLast()方法判断游标是否执行第一条记录或最后一条记录。
- Cursor 对象通过一组 getXXX()方法获取一条记录的各个字段的值。这组方法需要一个整数作参数，该整数就是字段的下标，从 0 开始。也可以通过 Cursor 对象的 getColumnCount()方法获取字段的数量。



图 7.3　查询操作结果

程序运行结果如图 7.3 所示。

当需要设置查询条件时，参数设置的方式和有条件更新或删除操作是一样的。例如，要查询 uname 值为“张三”的记录，query()方法的参数设置如下：

```
Cursor cursor = db.query(tableName, null, "uname = ?", new String[]
{ "张三" }, null, null, null, null);
```

如果要对 pwd 字段进行模糊查询，比如查询包含字符“3”的记录，那么 query 方法的参数设置如下：

```
Cursor cursor = db.query(tableName, null, "pwd like ?",new String[]{"%3%"},
null, null, null,null);
```

如果你觉得设置这些参数很麻烦，也不愿意拼接繁琐的 SQL 语句，SQLiteDatabase 对象提供了另外一种类似于 java.sql.PreparedStatement 的查询方式，rawQuery()方法，代码如下：

```
String sql="select * from users where uname=? and pwd like ?";
Cursor cursor = db.rawQuery(sql, new String[]{"张三","%3%"});
```

代码说明如下：

rawQuery()方法有两个参数，第一个是 SQL 语句，其中用占位符表示数值；第二个字符串数组，依次替换 SQL 语句中的占位符。

7.4　SQLiteOpenHelper 的使用

Android SDK 还提供了一个数据库帮助类，SQLiteOpenHelper。它提供了一套自动执行的机制来帮助开发者创建、更新、打开数据库。下面来看一下如何继承使用 SQLiteOpenHelper，代码如下：

```
public class SQLiteDB extends SQLiteOpenHelper{

    public SQLiteDB(Context context, String name, CursorFactory factory,
            int version) {
        super(context, name, factory, version);
                                    //调用父类 SQLiteOpenHelper 类的构造方法
    }

    public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE IF NOT EXISTS tableOne "
                                    //创建数据库表 sql 语句
                + "(uname VARCHAR(50), pwd VARCHAR(50));";
        db.execSQL(sql);            //执行 sql 语句
    }

    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        String sql = "CREATE TABLE IF NOT EXISTS tableTwo "
                                    //创建数据库表 sql 语句
                + "(uname VARCHAR(50), pwd VARCHAR(50));";
        db.execSQL(sql);            //执行 sql 语句
    }

    public String showTable() {
        SQLiteDatabase db = this.getReadableDatabase();
                                    //获取 SQLiteDatabase 对象
        Cursor cursor = db.query("sqlite master", new String[] { "name" },
                "type = ?", new String[] { "table" }, null, null, null, null);
```

```
                                    //查询所有数据库表名字
        String tables = "";
        if (cursor.getCount() != 0) {//判断获取结果的记录条数
            cursor.moveToFirst();   //游标定位到第一条记录
            for (int i = 0; i < cursor.getCount(); i += 1) {
                tables = tables + cursor.getString(0) + "\n";
                            //获取表中每条记录的字段值信息，保存到 tables 变量中
                cursor.moveToNext();//游标指向到下一条记录
            }
        }
        return tables;
    }
}
```

代码说明：

- 继承 SQLiteOpenHelper 的子类必须要调用父类的构造方法。
- onCreate()方法会在第一次实例化类对象的时候自动调用，且自动创建数据库。可以重写这个方法，添加一些建表或初始化数值的操作。
- onUpgrade()方法只有在版本信息发生变化时才会被调用。可以将在变化版本时需要修改的内容添加在里面。版本信息由实例化类的对象时传入。

下面来看一下如何使用 SQLiteOpenHelper 类，代码如下：

```
public class MyOpenHelper extends Activity{
    private Button firstButton;
    private Button secondButton;
    private SQLiteDB dbHelper;
    private final String dbName = "helperdb";
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.layout);

        firstButton = (Button) findViewById(R.id.first);
                                    //实例化 Button 对象
        firstButton.setOnClickListener(new OnClickListener() {
                                    //为 Button 对象添加监听
            public void onClick(View v) {
                dbVersion(1);           //方法调用
            }
        });

        secondButton = (Button) findViewById(R.id.second);
                                    //实例化 Button 对象
        secondButton.setOnClickListener(new OnClickListener() {
                                    //为 Button 对象添加监听
            public void onClick(View v) {
                dbVersion(2);           //调用方法
            }
        });
    }

    public void dbVersion(int version){
        dbHelper=new SQLiteDB(this,dbName,null,version);
                                    //实例化 SQLiteDB 类对象
        showMessage(dbHelper.showTable()); //显示数据库表信息
    }
```

```
    public void showMessage(String msg){    //定义信息框显示数据库表信息
        new AlertDialog.Builder(MyOpenHelper.this).setTitle("Message")
        .setMessage(msg).setNegativeButton("确定",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog,int which) {
                }
        }).show();
    }
}
```

代码说明：

- ❑ 在两个按钮中都在实例化 SQLiteOpenHelper 类对象，不同的只有版本参数。
- ❑ “first”按钮中实例化 SQLiteOpenHelper 类对象的操作会自动创建数据库，并执行 onCreate()方法中的代码。
- ❑ “second”按钮中实例化 SQLiteOpenHelper 类对象的操作，因为版本参数的变化会执行 onUpgrade()方法中的代码。

程序运行效果如图 7.4 和图 7.5 所示。



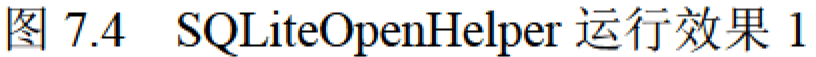

图 7.4　SQLiteOpenHelper 运行效果 1



图 7.5　SQLiteOpenHelper 运行效果 2

在 SQLiteOpenHelper 类中，可以执行前面章节中介绍的所有对 SQLite 数据库的操作。区别在于，在 SQLiteOpenHelper 类中需要通过下面的代码获取 SQLiteDatabase 类的对象：

```
SQLiteDatabase db = this.getReadableDatabase();
SQLiteDatabase db = this.getWritableDatabase();
```

另外，要完成对数据库内容写入/读出的操作，需要在应用程序中添加相应的权限：

```
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

第 8 章　多线程设计

伴随着多核 CPU 的诞生，使用多个线程进行编程是编程人员必须掌握的一门重要技术。多线程使用得当会使得程序运行更加快速，如果使用不得当，反而会适得其反。Java 本身就是支持多线程编程的开发语言，用其进行多线程的开发既方便又高效。本章主要介绍的内容是多线程开发的必知必会的知识，主要包括线程定义、启动方式、线程让步，以及同步。

8.1　多线程概述

在计算机编程中，一个基本的概念就是同时对多个任务加以控制。许多程序设计问题都要求程序能够停下手头的工作，改为处理其他一些问题，再返回主进程。可以通过多种途径达到这个目的。开始时，那些掌握机器低级语言的程序员编写一些“中断服务例程”，主进程的暂停是通过硬件级的中断来实现的。尽管这是一种有用的方法，但编出的程序很难移植，由此造成了另一类的代价高昂问题。中断对那些实时性很强的任务来说是很有必要的。但对于其他许多问题，只要求将问题划分进入独立运行的程序片断中，使整个程序能更加迅速地响应用户的请求。

多线程编程使得程序具有两条或者两条以上的并发执行线索，就像日常工作中由多人同时合作完成一个任务。在很多情况下，使用多线程可以改善程序的响应速率，提高资源利用率，这在多核 CPU 时代显得非常重要。但是有时也会因为滥用多线程后给程序带来意想不到的错误，降低执行效率。

在现实世界中有许多需要使用多线程的情况。例如，在网上注册用户后，一方面该网站会发给用户一封激活的邮件，同时也需要提醒用户注意接收。

如果是单线程模式，则需要等待邮件发送完成后显示提示信息，由于网络的因素可能导致邮件发送过程相对较慢，用户可能需要经过漫长的时间才可能收到，这样影响系统的性能。使用多线程模式，可以在发送邮件的时候，同时提醒用户注意接收邮件，这样提高了界面的响应速率，减少用户的等待时间。如图 8.1 所示。

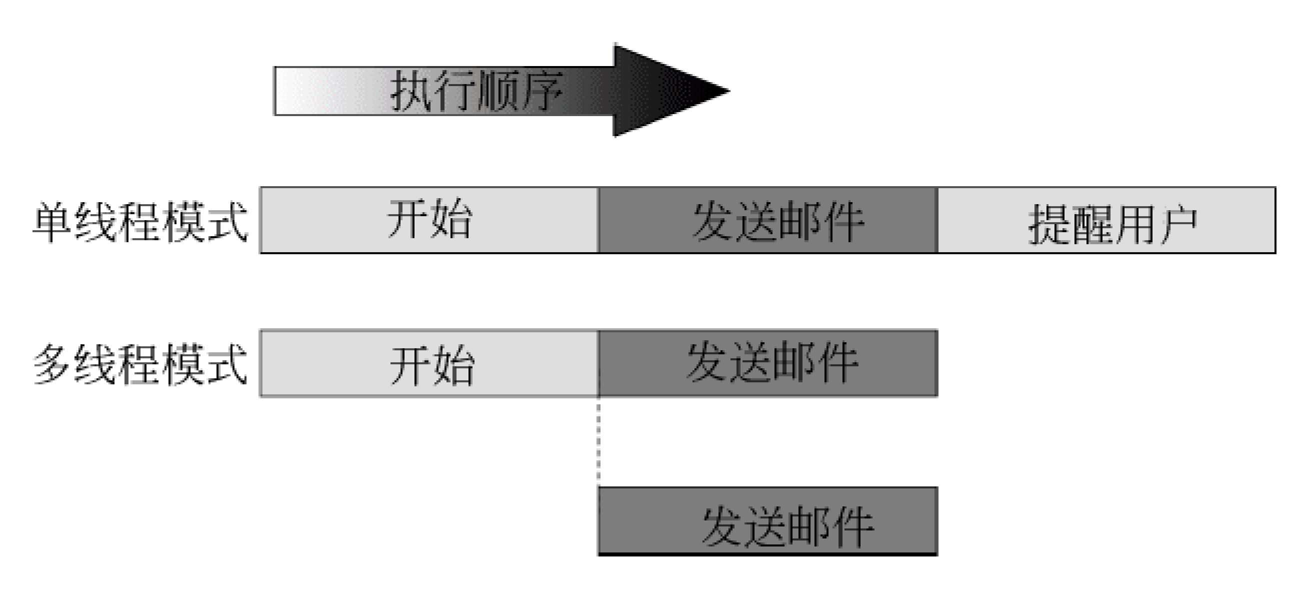


图 8.1　单线程与多线程

8.2　线程的启动方式 Thread

Java 中线程主要有两方面的含义，一是一条独立执行的线索，二是 java.lang.Thread 类或其子类的对象。在 Java 中开发自己的线程主要有两种方式，一种是继承自 Thread 类，另一种是实现 Runnable 接口，两种方式在不同的场合各有优缺点，读者可以在以后的开发中自己总结。

首先介绍继承 Thread 类的方式。如果一个类直接继承 Thread 类，则该类就是一个线程类，这是最基本的创建线程类的方法，采用此方式最重要的是继承 Thread 类后需要重写 run()方法。run()方法中存储了该线程所需要做的事情的描述。继承 Thread 类的基本语法如下。

```
class <类名> extends Thread
{
    @Override
    public void run()
    {
        //具体执行的代码
    }
}
```

代码说明：

- ❑ 在上述创建线程类的代码中，run()方法中编写的是该线程需要执行的具体代码，一旦该线程启动，run()方法将独立执行。
- ❑ 该类中的 run()方法也可以重载，只是重载之后该方法不再独立执行，在开发时读者千万要注意，不要写错。

下面给出了一个继承自 Thread 类的线程的例子，具体代码如下。

```
package pkg;
class MyThread extends Thread{                    //继承子 Thread 类
    @Override
    public void run(){                            //重写的 run()方法
        System.out.println("本类是通过继承 Thread 创建的线程");
    }
}
public class Sample8 1{
    public static void main(String[] args){     //main()方法
        MyThread mt=new MyThread();
        mt.start();
    }
}
```

代码说明：

- ❑ 在 run()方法中只是输出了“本类是通过继承 Thread 创建的线程”，其余并未做任何事情。读者可以根据自己的需要在 run()方法中写具体的代码。
- ❑ 在 main()方法中，首先创建 MyThread 类的对象，然后通过该对象调用 start()方法开启该线程。在开启线程时只能通过 Thread 类或其子类的对象调用 start()方法调用。

该案例的运行效果如图 8.2 所示。

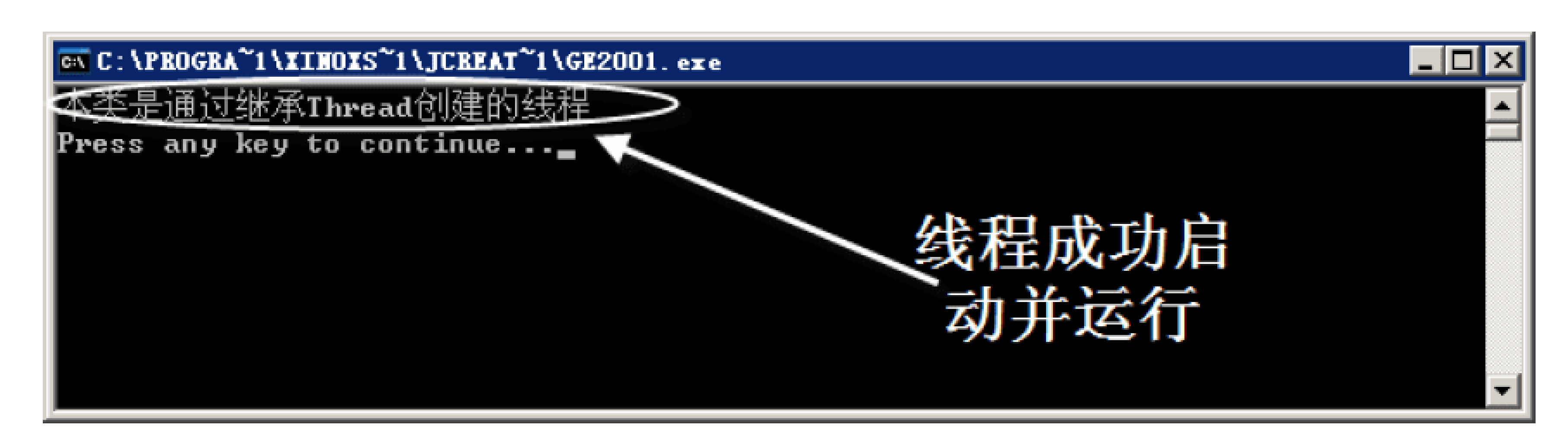


图 8.2　继承 Thread 类后创建的线程

8.3　线程的启动方式 Runnable

采用继承的方式开发的线程有一个较大的缺点，就是每次只能继承一个 Thread，并不可以像 VC++一样多继承。因此，Java 中提供了一种 Runnable(java.lang.Runnable)接口，实现该接口也可以创建线程类，但是在实现该接口之后仍需要实现其 run()方法。这样，实现 Runnable 接口之后同样也就具有了描述线程任务的 run()方法，此 run()方法也可以在一定条件下独立执行，其基本结构如下。

```
class <类名> implements Runnable
{
    @Override
    public void run()
    {
        //具体执行的代码
    }
}
```

下面给出了一个实现 Runnable 接口的线程的例子，具体代码如下。

```
package pkg;
class MyThread implements Runnable{
    @Override
    public void run(){
        System.out.println("本类是通过实现 Runnable 接口创建的线程");
    }
}
public class Sample8_2{
    public static void main(String[] args){
        MyThread mt=new MyThread();
        Thread td=new Thread(mt);
        td.start();
    }
}
```

代码说明：

- 在 run()方法中只是输入出了“本类是通过实现 Runnable 接口创建的线程”，其余并未做任何事情。读者可以根据自己的需要在 run()方法中写具体的代码。
- 在 main()方法中，首先创建 MyThread 类的对象，然后根据 Runnable 或其子类的对象创建 Thread 类的对象，最后通过 Thread 类的对象调用 start()方法。

该案例的运行效果如图 8.3 所示。



图 8.3　实现 Runnable 接口的线程

在第 8.2 节与 8.3 节中的代码可以看出，继承 Thread 的类创建线程对象的操作非常简单，而对于实现 Runnable 接口的类，其自身的对象并不是一个线程，只是在该类中通过实现 run()方法指出了线程需要的任务。然而，若想开启一个线程，必须创建 Thread 类或者其子类的对象，这时就需要特定的构造器来完成这个工作，Thread 类的常用构造器如表 8.1 所示。

表 8.1　Thread类的常用构造器

构造器签名	功　　能
public Thread()	该构造器将构造一个新的线程对象，该对象启动后将运行自身的 run()方法，并且该对象具有默认的名称
public Thread(Runnable rable)	参数 rable 为指定的 Runnable 实现类，该构造器将构造一个线程对象，当该对象启动后将执行指定的 rable 中的 run()方法，同样该对象也具有默认的名称
public Thread(Runnable rable,String name)	参数 rable 为指定的 Runnable 实现类，参数 name 为指定的名称，该构造器将构造一个新的线程对象，当该对象启动后将执行指定的 rable 中的 run()方法，该对象也具有指定的名称
public Thread(String name)	参数 name 为指定的名称，该构造器将构造一个新的线程对象，该对象启动后将运行自身的 run()方法，同样该对象也具有指定的名称

在 Thread 类的构造器列表中可以看出，当创建线程对象时，有时需要先创建实现 Runnable 接口的类的对象，然后将此对象的引用传递给 Thread 类构造器即可，这种方式实际上是告诉线程对象要执行的任务，即 run()方法在哪里。

8.4　线 程 休 眠

在前两节已经介绍了创建线程的具体方式方法，但其只设计到了单个线程，如果多个线程同时执行，其多个线程之间的先后顺序是怎样的，这个无法保障。因此 Java 提供了一些编程调度线程的方法，使用这些方法可以对线程在一定的程度上影响调度。但是读者需要注意，这些调度线程的方法，有些是有保障的，有些只是影响线程进入执行状态的概率，从微观角度考量是没有保障的。本节将简要地介绍最常用的线程调度方法——休眠。

在线程执行的过程中，调用 sleep()方法可以让线程睡眠一定的时间，等指定时间到达

之后，线程则会苏醒，并进入准备状态等待执行。这是使得正在执行的线程让出 CPU 的一种最简单也是最常用的方法。具体的方法如下：

```
public static void sleep(long millis) throws InterruptedException
public static void sleep(long millis,int nanos) throws InterruptedException
```

代码说明：

- 上述两个方法都可以使得线程进入睡眠状态，在睡眠一定时间后继续向下执行。
- 参数 millis 为指定线程将要睡眠的毫秒数，参数 nanos 为指定睡眠的纳秒数。但是纳秒是不准确的，不能作为时间基准。
- 上述两个方法中都可能出现 InterruptedException 捕获异常，所以在调用此方法时需要执行异常处理。
- 由于两个方法均是静态方法，所以这两个方法可以在任何位置通过 Thread 类或其子类直接调用。也就是说，哪个线程执行了 sleep()方法，则该线程进入休眠状态，并不是调用特定线程对象的 sleep()方法。

下面是一个两个线程调用 sleep()方法的例子，代码如下。

```
package pkg;
class runnable1 implements Runnable{
    public void run(){                              //重写的 run()方法
        for(int i=0;i<5;i++){
            System.out.println(""+i+" 我是实现 Runnable 接口的线程");
            try{
                Thread.sleep(100);                  //调用 sleep()方法
            }catch(Exception e) {
                e.printStackTrace();
            }
        }
    }
}
class runnable2 implements Runnable{
    public void run(){                              //重写的 run()方法
        for(int i=0;i<5;i++){
            System.out.println(""+i+" 我是实现 Runnable 接口的另一个线程");
            try{
                Thread.sleep(100);                  //调用 sleep()方法
            }catch(Exception e) {
                e.printStackTrace();
            }
        }
    }
}
public class Sample8_3{
    public static void main(String[] args){
        runnable1 mr1=new runnable1();
        runnable2 mr2=new runnable2();
        Thread t1=new Thread(mr1);
        Thread t2=new Thread(mr2);
        t1.start();                                 //开启线程
        try{
            Thread.sleep(5);                        //调用 sleep()方法
        }catch(Exception e) {
            e.printStackTrace();
        }
```

```
        t2.start();                                      //开启线程
    }
}
```

代码说明：

- ❑ 创建的第一个实现 Runnable 接口的类中的 run()方法为实现 Runnable 接口必须重写的方法。首先在该类中打印“i 我是实现 Runnable 的接口的线程”，然后调用 sleep()方法使得线程休眠，这时必须捕获异常。
- ❑ 创建的第二个实现 Runnable 接口的类中的 run()方法为实现 Runnable 接口必须重写的方法。首先在该类中打印“i 我是实现 Runnable 接口的另一个线程”，然后调用 sleep()方法使得线程休眠，但是调用 sleep()方法后必须捕获异常。
- ❑ 在类 Sample8_3 中的 main()方法中，首先创建 Runnable 类的对象，然后创建 Thread 类的对象，最后开启线程，同时调用 sleep()方法使得线程休眠，此时也需要捕获异常。

该案例的运行效果如图 8.4 所示。

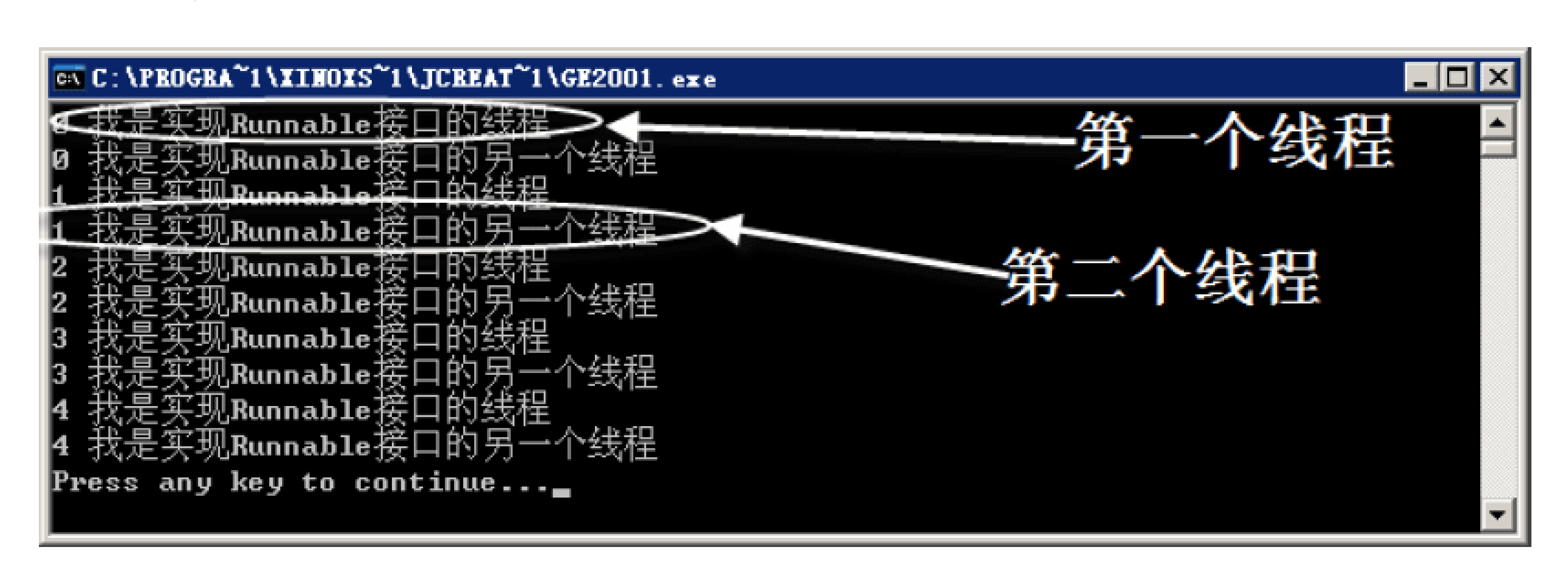


图 8.4　sleep()的方法

8.5　线 程 让 步

当线程等待另一个线程执行完毕才恢复执行时，可以使用 join()方法或者 yield()方法。顾名思义，使用 join()方法可以达到线程让步的效果。具体的 join()方法如下。

```
public final void join() throws InterruptedException
public final void join(long millis) throws InterruptedException
public final void join(long millis,int nanos) throws InterruptedException
```

- ❑ 上述 3 个方法均是 final 的方法，在继承 Thread 类时不能对其进行重写，同时，其有可能抛出 InterruptedException 捕获异常，因此在调用该方法时应该进行异常处理。
- ❑ 对于没有入口参数的 join()方法，将使得调用该方法的线程一直等待到此方法所在的线程执行完毕才恢复执行，效果上就好像两个线程合并为一个线程。
- ❑ 参数 millis 为指定等待的毫秒数，参数 nanos 为指定额外的纳秒数。对于没有入口参数的 join()方法，调用 join()方法的线程会等待相应的时间，如果在指定的时间内 join()方法所属的线程没有执行完毕则解除等待关系。

下面是一个使用 join()方法合并两个线程的例子，代码如下。

```
package pkg;
class runnable1 implements Runnable{
    @Override
    public void run(){                              //重写的 run()方法
        for(int i=0;i<=10;i++){
            System.out.println(""+i+"[runnable1]");
        }
    }
}
class runnable2 implements Runnable{
    @Override
    public void run(){                              //重写的 run()方法
        for(int i=0;i<=10;i++){
            System.out.println(""+i+"[runnable2]");     //输出
        }
    }
}
public class Sample8 4{
    public static void main(String[] args){     //main()方法
        runnable1 mr1=new runnable1();
        runnable2 mr2=new runnable2();
        Thread t1=new Thread(mr1);
        Thread t2=new Thread(mr2);
        t1.start();                                 //开启线程
        t2.start();                                 //开启线程
    }
}
```

代码说明：

- 创建的第一个实现 Runnable 接口的 runnable1 类中的 run()方法为实现 Runnable 接口必须重写的方法。在该类中主要是通过 for 循环打印“i[runnable1]”。
- 创建的第二个实现 Runnable 接口的 runnable2 类中的 run()方法为实现 Runnable 接口必须重写的方法。在该类中主要是通过 for 循环打印“i[runnable2]”。
- 在类 Sample8_4 中的 main()方法中，首先创建 Runnable 类的对象，然后创建 Thread 类的对象，最后开启线程。

该案例的运行效果如图 8.5 所示。

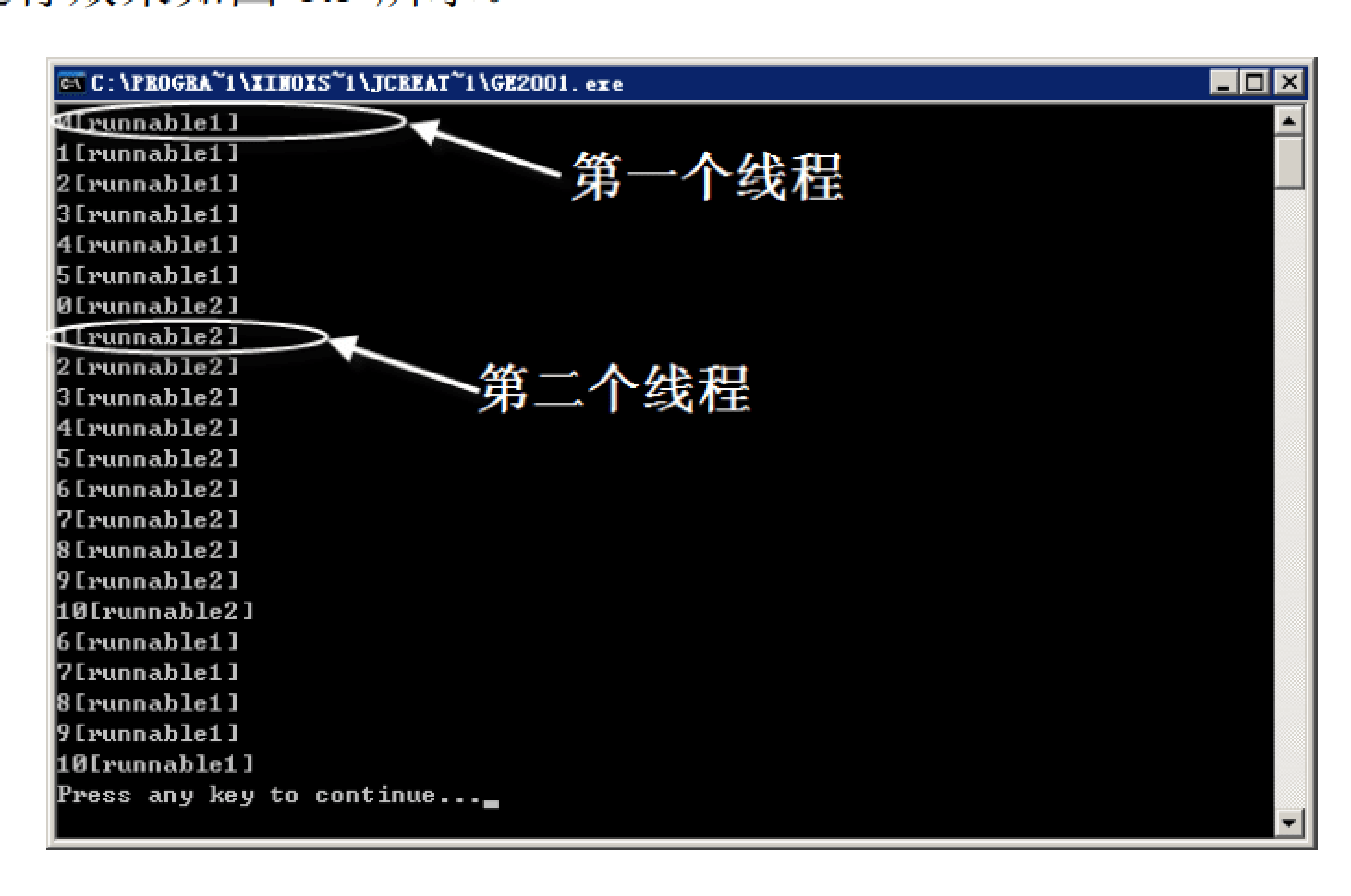


图 8.5　执行结果

调用 yield()方法也可以使得当前正在运行的线程让出 CPU，回到准备状态，进而使得其他线程有进入到运行态的机会。但是需要注意的是，该操作没有运行保障，很可能线程回到准备状态后又立刻被调度再次进入运行态，也就是说 yield()方法不一定能成功。其基本的方法结构如下：

```
public static void yield();
```

下面给出一个使用 yield()的例子，具体代码如下。

```
package pkg;
class MyRunnable implements Runnable{
    private String flagl;
    private String flagr;
    public MyRunnable(String flagl,String flagr){  //构造器
        this.flagl=flagl;
        this.flagr=flagr;
    }
    @Override
    public void run(){                             //重写的 run()方法
        for(int i=0;i<50;i++){                     //循环
            System.out.print(flagl+i+flagr);
            Thread.yield();
        }
    }
}
class MyThread extends Thread{
    private String str1;
    private String str2;
    public MyThread(String str1,String str2){      //构造器
        this.str1=str1;
        this.str2=str2;
    }
    @Override
    public void run(){                             //重写的 run()方法
        for(int i=0;i<50;i++){                     //循环
            System.out.print(str1+i+str2);
            Thread.yield();
        }
    }
}
public class Sample8 5{
    public static void main(String[] args){        //main()方法
        MyRunnable mr1=new MyRunnable("[","] ");
        MyThread t2=new MyThread("<","> ");
        Thread t1=new Thread(mr1);
        t1.start();                                //开启线程
        t2.start();
    }
}
```

代码说明：

- ❑ MyRunnable 类实现了 Runnable 接口，在该类中的构造器主要是初始化相应的数据，该类中重写了 run()方法，在该方法中主要是输出数据，并通过 Thread 类调用 yield()方法。
- ❑ MyThread 类继承了 Thread 类，在该类中的构造器主要是初始化相应的数据，该类中重写了 run()方法，在该方法中主要是输出数据，并通过 Thread 类调用 yield()

方法。

- ❑ 在类 Sample8_5 中主要是调用 main()方法，该方法首先创建相关类的对象，然后根据 Thread 类的对象调用 start()方法开启线程。

该案例的运行效果如图 8.6 所示。

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
[0] <0> [1] <1> [2] <2> [3] <3> [4] <4> [5] <5> [6] <6> [7] <7> [8] <8> [9] <9>
[10] <10> [11] <11> [12] <12> [13] <13> [14] <14> [15] <15> [16] <16> [17] <17>
[18] <18> [19] <19> [20] <20> [21] <21> [22] <22> [23] <23> [24] <24> [25] <25>
[26] <26> [27] <27> [28] <28> [29] <29> [30] <30> [31] <31> [32] <32> [33] <33>
[34] <34> [35] <35> [36] <36> [37] <37> [38] <38> [39] <39> [40] <40> [41] <41>
[42] <42> [43] <43> [44] <44> [45] <45> [46] <46> [47] <47> [48] <48> [49] <49>
Press any key to continue...
```

图 8.6　案例运行效果

8.6　线程的同步

多个线程中，由于同时有多个线程并发运行，这时可能会带来严重的问题。例如，一个银行账户在同一时刻只能由一个用户操作，不能两个用户同时对该账户进行操作。为了解决这个问题，就需要使用到多线程开发技术，在本节将对这方面的知识做些简要的介绍。

同步方法是指用 synchronized 关键字修饰的方法，其与普通方法不同的是进入同步方法执行的线程将获得同步方法所属对象的锁，一旦对象被锁，其他线程就不能执行被锁对象的任何同步方法。也就是说，线程在执行同步方法之前，首先试图获得方法所属对象的锁，如果不能获得锁就进入对象的锁等待池等待，直到别的线程释放锁，其获得锁才能执行。其基本语法如下：

```
synchronized <返回类型> 方法名称([参数列表]) [throws <异常序列>]
{
    //同步方法的方法体
}
```

方法说明：

- ❑ 关键字 synchronized 只能标识方法，不能标识成员变量。
- ❑ 一个对象可以同时有同步与非同步的方法，只有进入同步方法执行才需要获得锁，每个对象只能有一个锁。如果一个对象中有同步方法，则某线程访问该方法时，其他线程不能访问该方法，只能等待上一线程访问完毕之后才可以。
- ❑ 如果线程获得锁后进入睡眠或者进行让步，则将带着锁一起睡眠或让步，这种做法将会严重影响等待锁线程的执行，进而影响程序的整体性能。
- ❑ 同步方法降低了程序的并发性，但同时也在一定程度上影响了系统的性能。

下面是一个使用 synchronized()方法的例子。

```
package pkg;
class Resource{
    synchronized void function(Thread currThread){ //创建同步方法
        System.out.println(currThread.getName()+
        "线程执行 synchronized 定义的方法。");
        try{
            Thread.sleep(2000);                    //线程休息
```

```
                System.out.println(currThread.getName()
                +"线程睡醒了。");
            }catch(Exception e){
                e.printStackTrace();                        //打印堆栈信息
            }
        }
}
class MyThread extends Thread{
    Resource res;
    String tName;
    public MyThread(String tName,Resource res){     //构造器
        this.setName(tName);
        this.tName=tName;
        this.res=res;
    }
    public void run(){
        if(tName.equals("Thread1")){
            res.function(this);                         //调用同步方法
            System.out.println("Thread2 启动，进入同步方法中");
        }
    }
}
public class Sample8_6{
    public static void main(String args[]){
        Resource rs=new Resource();                     //创建 Resource 类的对象
        MyThread t1=new MyThread("Thread1",rs);
        MyThread t2=new MyThread("Thread2",rs);
        t1.start();
         try {
            Thread.sleep(10);                           //线程休息
         }catch(Exception e){
            e.printStackTrace();                        //打印堆栈信息
         }
        t2.start();
    }
}
```

代码说明：

- 自定义的 Resource 类中的 synchronized 修饰的方法为同步方法，该方法在同一时刻只能被同一线程访问。在该方法中主要是打印相关的进度信息。
- 自定义的 MyThread 类中的 MyThread(String tName,Resource res)方法是该类的构造器，在该构造器中主要是设置名称并初始化相应的数据。
- 自定义的 MyThread 类继承自 Thread 类，所以需要重写 run()方法，在该方法中主要是判断传入的参数是否为“Thread1”。
- 在 Sample8_6 类中的 main()方法中主要是创建了线程类 MyThread 类的对象，并通过该对象启动相关的线程。

该示例的运行效果如图 8.7 所示。



图 8.7　使用 synchronized()方法的示例运行效果

在前面介绍了 synchronized 修饰的同步方法，并且介绍了该 synchronized 关键字的使用，线程在推出该同步方法后会释放方法所属对象的锁。但这并不是全部，在同步方法中还可以使用特定的方法对线程进行调度，这些方法在处理资源的同步时非常有效，其中主要的方法如表 8.2 所示。

表 8.2　同步方法中调用线程的方法

方 法 名 称	具 体 功 能
public final void wait() throws InterruptedException	该方法将使得某一线程进入资源的等待池，使其进入等待状态，直至别的线程调用该资源的 notify()或者 notifyAll()方法将其唤醒为止。该方法可能抛出 InterruptedException
public final void wait(long timeout) throws InterruptedException	参数 timeout 为指定的毫秒数，该方法将使得某一线程进入该资源的等待池，使其进入等待状态，直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者超过指定的时间量。该方法可能抛出 InterruptedException
public final void wait(long timeout,int nanos) throws InterruptedException	参数 timeout 为指定等待的毫秒数，参数 nanos 为指定额外的纳秒数。该方法将使得某一线程进入该资源的等待池，使其进入等待状态，直到其他线程调用此对象的 notify() 方法或 notifyAll()方法，或者超过指定的时间量。该方法可能抛出 InterruptedException
public final void notify()	唤醒在此对象监视器上等待的单个线程。如果所有线程都在此对象上等待，则会选择唤醒其中一个线程。选择是任意性的，并在对实现做出决定时发生。线程通过调用其中一个 wait()方法，在对象的监视器上等待
public final void notifyAll()	唤醒在此对象监视器上等待的所有线程。线程通过调用其中一个 wait()方法，在对象的监视器上等待

在资源的同步与互斥问题中，最典型的就是“生产者——消费者”的问题了。其具体的含义是，系统中有很多生产者和消费者同时工作，生产者负责生产需要的资源，消费者消耗资源。当消费者消费资源时，如果资源不足，则需要等待，反之当生产者生产资源时，如果资源已经满足需要，则也需要等待。并且同一时刻只能有一个生产者或者消费者进行操作。

下面给出一个生产者与消费者的具体示例，代码如下。

```
package pkg;
class BreadContainer{
    public static final int maxNum=300;
    private int num;
    public BreadContainer(){    }
    public BreadContainer(int num) {                //有参构造器
        this.num=num;
    }
    public synchronized void produceBread(int produceNum,String
```

```
    producerName)   {
        while(num+produceNum>maxNum){
            System.out.println(producerName+"要生产"+produceNum+"个，当前"
            +num+"个，资源充足，不需要生产，"+producerName+"去等待！");
            try{
                wait();                                    //等待
            }catch(Exception e){
                e.printStackTrace();                       //打印堆栈信息
            }
        }
        num=num+produceNum;
        System.out.println(producerName+"生产了"
                        +produceNum+"个，现在有"+num+"个。");
        notifyAll();                                       //调用 notifyAll()方法
    }
    public synchronized void consumeBread(int consumeNum,String consumerName){
        while(consumeNum>num)   {
            System.out.println(consumerName+"要消费"+consumeNum+
                "个，由于现在只有"+num+"个，"+consumerName+"于是去等待！");
            try{
                wait();
            }catch(Exception e){
                e.printStackTrace();
            }
        }
        num=num-consumeNum;
        System.out.println(consumerName+"消费了"
             +consumeNum+"个，现在还剩下"+num+"个");
        this.notifyAll()                               //调用 notifyAll()方法
    }
}
class Producer extends Thread{
    private int produceNum;
    private BreadContainer bc;                         //成员变量
    public Producer(){  }
    public Producer(int produceNum,BreadContainer bc,String producerName){
                                                       //有参构造器
        this.produceNum=produceNum;
        this.bc=bc;
        this.setName(producerName);
    }
    public void run(){                                 //重写的方法
        bc.produceBread(produceNum,this.getName());
    }
}
class Consumer extends Thread{
    private int consumeNum;
    private BreadContainer bc;
    public Consumer(){  }
    public Consumer(int consumeNum,BreadContainer bc,String consumerName){
        this.consumeNum=consumeNum;
        this.bc=bc;
        this.setName(consumerName);
    }
    public void run(){                                 //重写的方法
        bc.consumeBread(consumeNum,this.getName());
    }
}
public class Sample8_7{
```

```
    public static void main(String args[]){         //main()方法
        BreadContainer bc=new BreadContainer(50);
        Producer p1=new Producer(50,bc,"P1");
        Producer p2=new Producer(200,bc,"P2");
        Producer p3=new Producer(290,bc,"P3");
        Consumer c1=new Consumer(70,bc,"c1");
        Consumer c2=new Consumer(80,bc,"c2");
        c1.start();                                 //开启线程
        c2.start();
        p1.start();
        p3.start();                                 //开启线程
        p2.start();
    }
}
```

代码说明：

- 在 BreadContainer 类中的构造器主要有无参构造器与有参构造器，在有参构造器中主要是初始化相应的成员变量。
- 在 BreadContainer 类中主要声明了 produceBread()方法与 consumeBread()方法，这两个方法分别是生产者需要调用的方法与消费者调用的方法。在这两个方法最后是调用的 notifyAll()方法，调用该方法使得所有的线程均开启。
- Producer 类是继承 Thread 的线程类，该类中主要是重写了 run()方法，该方法中主要是调用 BreadContainer 类中的 produceBread()方法。
- Consumer 类是继承 Thread 的线程类，该类中主要是重写了 run()方法，该方法中主要是调用 BreadContainer 类中的 consumeBread()方法。
- 在类 Sample8_7 中主要是调用 main()方法，该方法中创建了 BreadContainer 类、Producer 类，以及 Consumer 类的对象，然后根据这些类的对象调用 start()方法开启线程。

该示例的运行效果如图 8.8 所示。

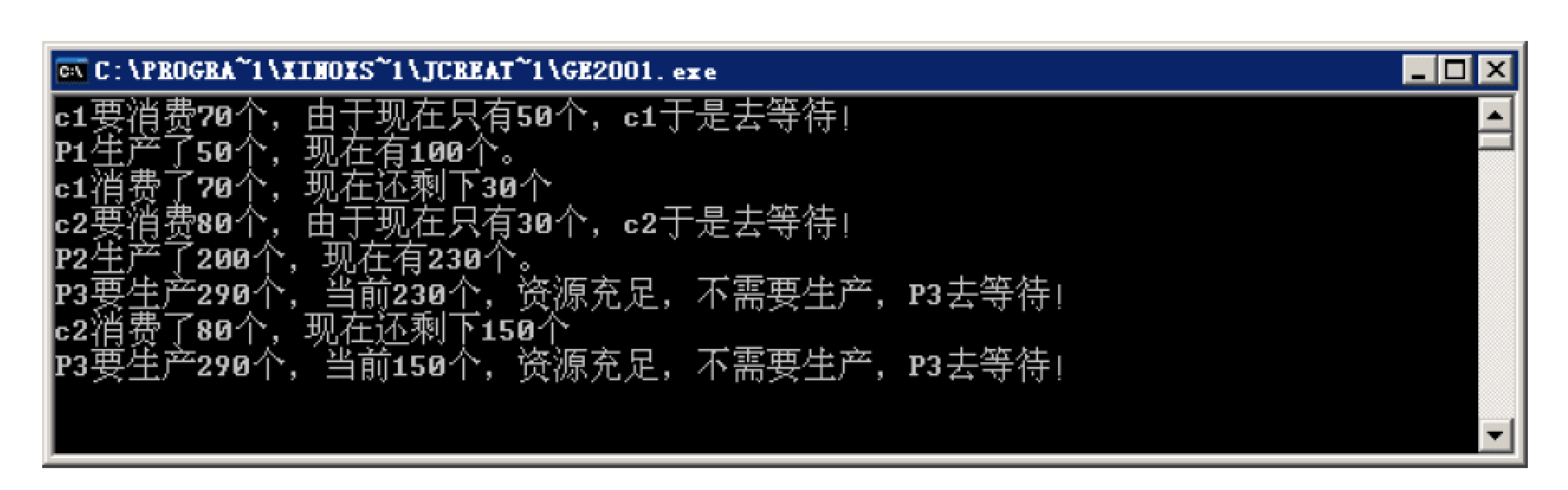


图 8.8　生产者——消费者

8.7　Android 中的 Service

手机不是电脑，其在同一时刻只能有一个窗口，如果在同一时刻需要开启多个窗口，则需要隐藏的窗口具有后台运行的能力。例如，在听歌曲的同时可以浏览网页，播放歌曲的软件必须要有后台运行的能力。

Android 平台在设计的过程中已经充分考虑到了这一点，Android 平台后台运行任务用的是 Service，而后台的 Service 可以通过 BroadcastReceiver 来响应其他组件发送的 BroadcastIntent，从而达到实现前台 Activity 与后台 Service 的交互。

创建自己的 Service 必须继承系统的 android.app.Service 类，然后重写 Service 生命周期状态，迁移过程中系统要回调各个方法。Service 的生命周期回调的方法主要有 6 个，分别是 onCreate()、onStart()、onDestory()、onBind()、onUnbind()，以及 onRebind()。下面是一个使用 Service 更换应用程序内容的示例，代码如下。

主控制类 SampleActivity.java。

```
package com.WindowExample;
import android.app.Activity;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
import android.widget.TextView;
public class SampleActivity extends Activity{
    static TextView tv;
    static Button button;                              //成员变量
   @Override
   public void onCreate(Bundle savedInstanceState){//重写的 onCreate()方法
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);                  //跳转页面
       button=(Button)findViewById(R.id.Button01);
       tv=(TextView)findViewById(R.id.textview01); //获得对象
       button.setOnClickListener(
        new OnClickListener(){                         //设置监听
                @Override
                public void onClick(View v){           //重写的方法
                    Intent intent=new Intent(SampleActivity.this,
                    MyService.class);
                   startService(intent);
             }
        }
       );
   }
   @Override
   public boolean onKeyDown(int KeyCode,KeyEvent event){  //重写的方法
        if(KeyCode==4){
            System.exit(0);
        }
        return true;
   }
}
```

代码说明：

- 该类继承自 Activity，重写了 onCreate()方法。该方法为程序开始时执行的方法，在该方法中主要是对 Button 按钮设置监听，点击该按钮后，创建 Intent 对象并开启 Service。
- onKeyDown()为重写的按键监听的方法，如果点击的是返回键，则调用 System.exit(0)。

Service 类 MyService.java。

```
package com.WindowExample;
import android.app.Service;
import android.content.Intent;                     //导入相关包
```

```
import android.os.IBinder;
public class MyService extends Service{
    static final String action1="Broadcast action1";   //定义字符串
    Intent it;
    @Override
    public IBinder onBind(Intent intent){        //重写的方法
        return null;                             //由于用不到，所以返回 null
    }
    @Override
    public void onCreate(){                      //重写的 onCreate()方法
        super.onCreate();
        new Thread(){                            //创建线程
            @Override
            public void run(){                   //重写的方法
                while(true){
                    it = new Intent(action1);
                    sendBroadcast(it);
                    try{
                        Thread.sleep(200);       //休眠
                    }catch(Exception e){
                        e.printStackTrace();     //打印信息
                    }
            } }
        }.start();                               //开启线程
    }
    @Override
    public void onDestroy(){                     //重写的 onDestroy()
        super.onDestroy();
        this.stopService(it);                    //停止 Service
    }
}
```

代码说明：

- 该类集成在 Service，并重写了 onBind()方法，由于该方法用不到，所以返回 null。在该类中还重写了 onCreate 方法()，在该方法中创建 Intent 对象，并调用 sendBroadcase 发送 Intent 的对象，并休眠。
- 重写了 onDestroy()方法，在该方法中调用了父类的 onDestroy，并调用 stopService 停止 Service。

接收类 CommandReceiver.java。

```
package com.WindowExample;
import android.content.BroadcastReceiver;                    //导入相关包
import android.content.Context;
import android.content.Intent;
public class CommandReceiver extends BroadcastReceiver {
   int status;                                               //状态值
   public static final String UPDATE_STATUS="UPDATE";        //常量字符串
    @Override
    public void onReceive(final Context context, Intent intent) {
                                                 //重写的 onReceive()方法
        updateUI(context);
    }
    public void updateUI(Context context){       //自定义的 updateUI()方法
        try{
            SampleActivity.tv.setTextSize(30); //设置字体大小
```

```
            SampleActivity.tv.setText(ZMDUtil.next());      //设置字体
        }catch(Exception e){
        }
    }
}
```

代码说明：

- ❑ CommandReceiver 类继承自 BroadcaseReceiver，在该类中主要是重写了 onReceive()方法，该方法主要是调用自定义的 updateUI()方法。
- ❑ updateUI()方法为自定义的更新 UI 界面的方法，在该方法中可以设置字体大小，以及设置字体。

工具类 ZMDUtil.java。

```
package com.WindowExample;
public class ZMDUtil {
    static int currIndex=0;                             //索引值
    public static String[] MSG={                        //字符串一维数组
        "德国大众：“小即是好。” ",
        "可口可乐：“享受清新一刻。” ",
        "万宝路香烟：“万宝路的男人。” ",
        "耐克：“说做就做。”",
        "麦当劳：“你理应休息一天。” ",
        "通用电气：“GE 带来美好生活。” ",
        "桌张频酒：“美妙口味不可言传。” ",
        "克莱罗染发水：“她用了？她没用？” ",
        "艾维斯：“我们正在努力。” ",
        "美国联邦快递公司：“快腿勤务员。” ",
        "阿尔卡-舒尔茨公司：“多种广告”。 ",
        "百事可乐：“百事，正对口味。” ",
        "麦氏咖啡：“滴滴香浓，意犹未尽。” ",
        "美国捷运公司：“你知道我吗？” ",
        "美国征兵署：“成为一个全材。” ",
        "Anacin 去痛片：“快、快、快速见效。” ",
        "滚石乐队：“感觉是真实的。” ",
        "百事可乐：“新一代的选择。” ",
        "哈斯维衬衫：“穿哈斯维的男人。” ",
        "博马剃须刀：“公路道边的招牌阵。” ",
        "美国汉堡王：“带着它上路。” ",
        "迪比尔斯：“钻石恒久远，一颗永留传。” "
    };
    public static String next(){                        //自定义的 next()方法
        String result=MSG[currIndex];
        currIndex=(currIndex+1)%MSG.length;
        return result;                                  //返回
    }
}
```

代码说明：

- ❑ 该类中 currIndex 为相应的一维数组的索引值。MSG 为一维数组，该数组中的数据是需要被显示的。
- ❑ ZMDUtil 类中 next()方法是动态的查找数组下一个的方法。

配置文件 main.xml。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
     <TextView
         android:id="@+id/textview01"
         android:textSize="20dip"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Test marquee for TextView"
        android:layout_gravity="center"
        android:singleLine="true"
    />                                              <!--TextView-->
     <Button android:text="点击开启 Service"
         android:id="@+id/Button01"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content">
     </Button>                                      <!--Button-->
</LinearLayout>
```

代码说明：

- 上述配置文件中声明了 TextView 的配置，声明了 TextView 的 ID、宽度、高度、显示的文字、所占位置，以及是否为单行。
- 上述配置文件中声明了 Button 按钮的配置，声明了按钮的 ID、大小，以及名称。

配置文件 AndroidManifest.xml。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.WindowExample"
     android:versionCode="1"
     android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/
    app_name">
       <activity android:name=".SampleActivity"
                android:label="@string/app_name">
          <intent-filter>
             <action android:name="android.intent.action.MAIN" />
             <category android:name="android.intent.category.LAUNCHER" />
          </intent-filter>
       </activity>
        <service android:name=".MyService"/>
        <receiver android:name=".CommandReceiver">    <!-- 注册接收器 -->
          <intent-filter>
            <action android:name="Broadcast_action1" />
          </intent-filter>
        </receiver>
    </application>
    <uses-sdk android:minSdkVersion="7" />
</manifest>
```

代码说明：

- 通过<service android:name=".MyService"/>注册 Service 组件。
- 通过<receiver android:name=".CommandReceiver">注册接收器，通过<action android:name="Broadcast_action1" />注册事件。

- 通过<uses-sdk android:minSdkVersion="7" />声明分辨率。

该示例的运行效果如图 8.9 和图 8.10 所示。



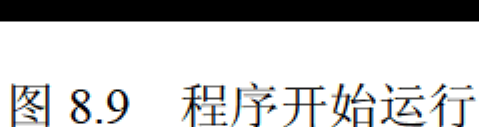

图 8.9　程序开始运行



图 8.10　点击按钮后

8.8　使用 Handler

在 Android 中可以通过 Handler 为发送消息，当消息队列有消息时，有消息循环的线程就可以一次处理消息队列中的消息，需要注意的是，这里的队列是符合 FIFO 的。Handler 通过 sendEmptyMessage 发送消息给系统，系统判断消息的类型，然后根据类型执行不同的动作。下面是一个使用 Handler 的示例，代码如下。

主控制类 SampleActivity.java 的代码。

```
package com.WindowsSample;
import android.app.Activity;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
import android.widget.Button;                              //导入相关类
import android.widget.TextView;
public class SampleActivity extends Activity{
    Handler hd=new Handler(){
        @Override
        public void handleMessage(Message msg) {           //重写方法
            switch(msg.what){
                case 0:
                    SampleActivity.tv.setTextSize(20); //设置字体大小
                    SampleActivity.tv.setText(ZMDUtil.next()); //设置字体
                break;
            }
        }
    };
    static TextView tv;
    static Button button;                                  //成员变量
   @Override
```

```
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        button=(Button)findViewById(R.id.Button01);      //创建 Button 对象
        tv=(TextView)findViewById(R.id.textview01);      //创建 TextView 对象
        button.setOnClickListener(                       //设置监听
         new OnClickListener(){
                  @Override
                  public void onClick(View v){           //重写的方法
                      new Thread(){
                      @Override
                      public void run(){                 //重写的方法
                          while(true) {
                              SampleActivity.this.hd.sendEmptyMessage(0);
                          }
                      }
                     }.start();                          //开启线程
              }
        } );
    }
}
```

代码说明：

- ❑ 在主控制类 SampleActivity 中创建了 Handler 的对象，并判断 msg 的值，如果值为 0，则设置字体大小为 30，并通过 ZMDUtil 调用 next()方法。
- ❑ 在主控制类 SampleActivity 中声明了静态的成员变量 tv 与 button。
- ❑ 主控制类 SampleActivity 中的 onCreate()方法为程序开始执行的方法，在该方法中首先调用父类，然后设置显示的页面，并得到 TextView 与 Button 的对象，最后对 Button 对象设置监听，如果点击该按钮调用 Handler 的对象，发送消息 0。

自定义类 ZMDUtil.java 的代码。

```
package com.WindowsSample;
public class ZMDUtil {
    static int currIndex=0;                           //索引值
     public static String[] MSG={                     //字符串一维数组
         "百威啤酒：“这百威是给你的。”",
         "维克多语言机器公司：“大师级的声音。” ",
         "木莓香皂：“光洁皮肤，不禁触摸。”",
         "本森.贺杰斯 100 周年：“我们的缺点。”",
         "全国饼干公司：“UNEEDA BISCUITS’BOY IN BOOTS。”",
         "劲量电池：“劲量兔子。”",
         "香奈尔香水：“分享这份梦幻。” ",
         "福特汽车“土星”系列：“不一样的公司，不一样的汽车。” ",
         "佳洁士牙膏：“看，妈妈，没有蛀牙。”",
         "玛氏巧克力：“只溶在口，不溶在手。”",
         "雪佛兰汽车：“开着你的雪佛兰看美国。” ",
         "云丝顿烟草：“云丝顿，好烟的好品味。” ",
         "骆驼香烟：“为了买这包骆驼香烟，我走了一英里。”",
         "凯迪拉克汽车：“做领袖 某头！！?",
         "小麦一族：“冠军的早餐。”",
         "可口可乐：“真正可口可乐。”",
         "灰狗长途汽车公司：“只有坐车之趣，没有驾车之累。” ",
         "宝丽莱即拍即得：“就是这么简单。” ",
```

```
        "克勒格大米咖哩：“咬一口，干干脆。”  ",
        "吉列剃刀：“看着光，感觉爽。”  ",
        "好运香烟：“只为好运，  不要甜蜜。”  ",
        "七喜汽水：“这不是可乐。”"
    };
    public static String next(){                        //自定义的 next()方法
        String result=MSG[currIndex];
        currIndex=(currIndex+1)%MSG.length;
        return result;                                  //返回
    }
}
```

代码说明：

- ZMDUtil 类中 currIndex 为相应的一维数组的索引值。
- ZMDUtil 类中 MSG 为一维数组，该数组中的数据是需要被显示的。
- ZMDUtil 类中 next()方法是动态的查找数组下一个的方法。

配置文件 main.xml 的代码。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:id="@+id/textview01"
        android:textSize="20dip"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Test marquee for TextView"
        android:layout_gravity="center"
        android:singleLine="true"
        android:focusable="true"
        android:marqueeRepeatLimit="marquee_forever"
        android:focusableInTouchMode="true"
        android:scrollHorizontally="true"
    />                                              <!--TextView-->
    <Button android:text="点击开启 Handler"
        android:id="@+id/Button01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
    </Button>                                       <!--Button-->
</LinearLayout>
```

代码说明：

- 上述配置文件中声明了 TextView 的配置，声明了 TextView 的 ID、宽度、高度、是否为单行、显示的文字，以及其所占位置。
- 上述配置文件中声明了 Button 按钮的配置，声明了按钮的 ID、大小，以及名称。

该示例的运行效果如图 8.11 和图 8.12 所示。

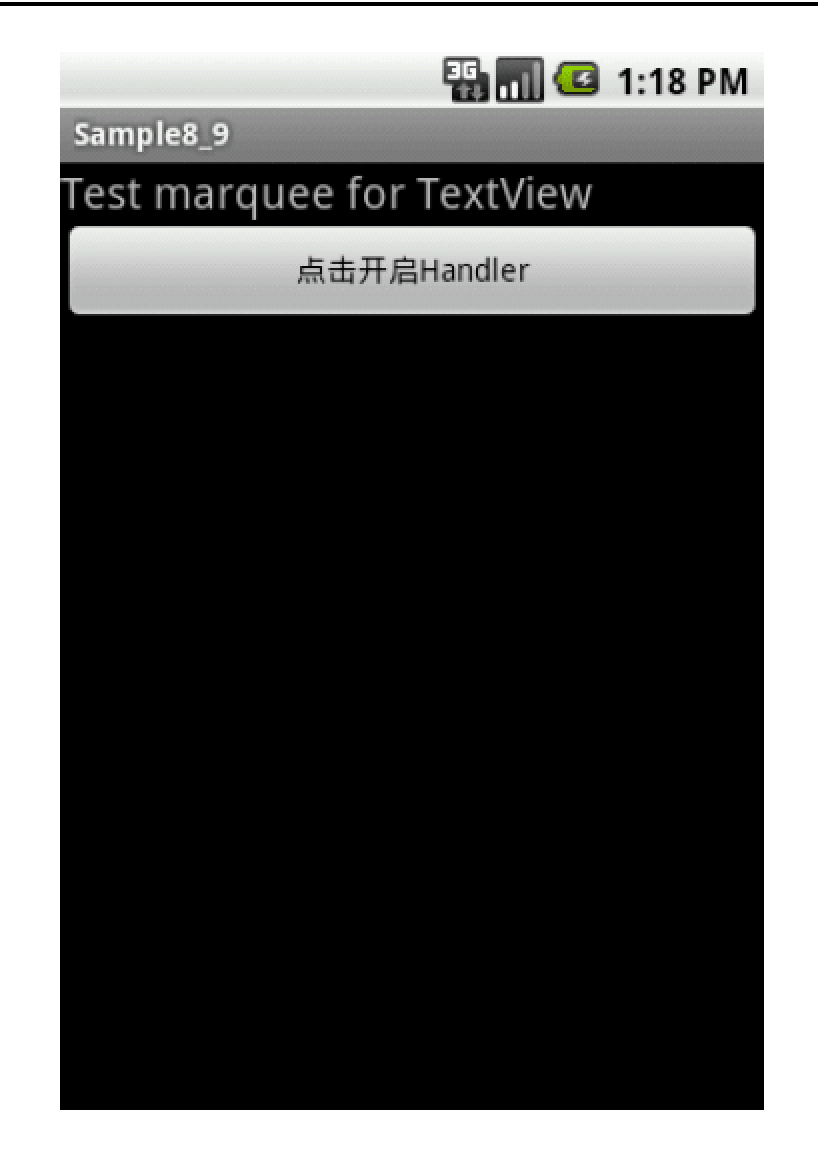


图 8.11　开始界面



图 8.12　点击按钮之后

8.9　使用 Looper

Handler 和 Thread 不一定是一一对应的。理论上，在一个 LooperThread 中，可以有多个 Handler，每个消息都可以指定不同的 Handler，因此每个消息都有不同的行为。

Looper 用于封装了 Android 线程中的消息循环。默认情况下一个线程是不存在消息循环（message loop）的，需要调用 Looper.prepare()方法来给线程创建一个消息循环，调用 Looper.loop()方法来使消息循环起作用，从消息队列里取消息，处理消息。

写在 Looper.loop()之后的代码不会被立即执行，当调用 mHandler.getLooper().quit()后，loop 才会中止，其后的代码才能得以运行。Looper 对象通过 MessageQueue 来存放消息和事件。一个线程只能有一个 Looper，对应一个 MessageQueue。下面是一个使用 Looper 的具体示例，代码如下。

主控制类 SampleActivity.java。

```
package com.WindowsSample;
import android.app.Activity;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
import android.widget.Button;                                    //导入相关类
import android.widget.TextView;
public class SampleActivity extends Activity{                    //创建主控制类
    Handler hd=new Handler(){
        @Override
        public void handleMessage(Message msg) {                 //重写方法
            switch(msg.what) {
                case 0:
                    SampleActivity.tv.setTextSize(20); //设置字体大小
                    SampleActivity.tv.setText(ZMDUtil.next()); //设置字体
                break;
            }
        }
```

```
    };
    static TextView tv;
    static Button button;                                        //成员变量
    static Lopper looper;
   @Override
   public void onCreate(Bundle savedInstanceState) {   //onCreate()方法
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);                            //跳转页面
       button=(Button)findViewById(R.id.Button01);
       tv=(TextView)findViewById(R.id.textview01); //获得 TextView 的对象
       looper=new Lopper(this);
       button.setOnClickListener (                          //设置监听
        new OnClickListener(){
               @Override
               public void onClick(View v){            //重写的方法
                   looper.start();
               }
       } );
   }
}
```

代码说明：

- 在主控制类 SampleActivity 中首先创建了 Handler 对象；然后调用 switch()方法判断 msg 的值，如果值为 0，则设置字体大小为 30；最后通过 ZMDUtil 调用 next()方法设置显示字体。
- 在主控制类 SampleActivity 中声明了 TextView、Button，以及 Lopper 成员变量。
- 主控制类 SampleActivity 中的 onCreate()方法为案例开始运行时首先执行的方法，在该方法中首先跳转页面；然后获得 TextView、Button()以及 Lopper 的对象；最后为 Button 按钮设置监听，如果点击该按钮，则开启线程 Lopper。

线程类 Lopper.java。

```
package com.WindowsSample;
import android.os.Looper;                                   //导入相关类
public class Lopper extends Thread {
    SampleActivity spa;
    boolean flag=true;                                      //flag 标志位
    public Lopper(SampleActivity spa){
        this.spa=spa;                                       //初始化
    }
    @Override
   public void run() {                                      //重写的 run()方法
       Looper.prepare();
       while(flag){
        spa.hd.sendEmptyMessage(0);                         //发送消息
       }
       Looper.loop();                                       //调用 loop()方法
       spa.button.setText("更改 Button 的名称");             //更改名称
   }
}
```

代码说明：

- 在线程类 Lopper 中声明了主控制类 SampleActivity 的引用，以及循环标志位。
- 线程类 Lopper 中的构造器主要是初始化成员变量获得主控制类 SampleActivity 的

对象。

- ❑ 本类中的 run()方法为继承 Thread 类需要重写的方法，在该方法中首先调用 prepare()方法，然后执行 while 循环，在该循环中不断地发送消息，最后调用 loop()方法，并更改 Button 按钮的名称，但是该更改名称并不能成功。

自定义类 ZMDUtil.java 的代码。

```
package com.WindowsSample;
public class ZMDUtil {
   static int currIndex=0;                                //索引值
    public static String[] MSG={                          //字符串一维数组
        "伟斯科清洁剂：“请涂在领子上。” ",
        "生活谷物：“你好，麦基。”",
        "赫特兹汽车租赁公司：“让赫特兹带你上路。” ",
        "颇度肉鸡：“让一个强硬的男人做一只松软的香鸡。”",
        "豪马克（英国伦敦金业工会）：“至诚关怀，真金表达。”",
        "格林斯宝罗集团：“杰克森高地公寓。”",
        "斯特恩威钢琴：“不朽的乐器。”",
        "布来克格拉马大湖皮草：“是什么活在传奇里？”",
        "ESPN 体育频道：“这里是体育中心。” ",
        "加州牛奶促进委员会：“喝牛奶了吗？”",
        "布莱尔克里姆护发乳：“每次只用一点点。”",
        "卡灵黑标啤酒：“嘿，梅宝来瓶黑标。” ",
        "德士古石油公司：“把你的车托给这颗星，你尽可放心。”",
        "施乐复印机：“这是一个奇迹。” ",
        "巴托斯与乔伊斯酒品冷却器：“弗兰克和艾迪”民俗二重唱。 ",
        "沃尔沃汽车:“在瑞典,一辆普通汽车的生涯。”",
        "6 字汽车旅馆连锁店：“我们为你留着一盏灯。” ",
        "吉尔-0 餐厅甜点：“比尔考斯比与孩子们。”",
        "梅宝即食早餐：“今天我 40 岁了，我要我的梅宝。”",
        "箭牌衬衫：“我的朋友，乔.赫尔姆箕斯，现在是一匹马。” ",
        "杨.罗比坎姆广告公司：“冲击。” ",
        "国际商用机器公司：“卓别林的小流浪形象。”"
   };
    public static String next(){                          //自定义的 next()方法
        String result=MSG[currIndex];
        currIndex=(currIndex+1)%MSG.length;
        return result;                                     //返回
    }
}
```

代码说明：

- ❑ ZMDUtil 类中 currIndex 为相应的一维数组的索引值。
- ❑ ZMDUtil 类中 MSG 为一维数组，该数组中的数据是需要被显示的。
- ❑ ZMDUtil 类中 next()方法可以查找数组的下一个数据。

配置文件 main.xml。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout width="fill parent"
   android:layout height="fill parent"
   >
    <TextView
```

```
        android:id="@+id/textview01"
        android:textSize="20dip"
       android:layout width="fill parent"
       android:layout height="wrap content"
       android:text="Test marquee for TextView"
       android:layout gravity="center"
       android:singleLine="true"
   />                                          <!--TextView-->
    <Button android:text="点击开启 Looper"
        android:id="@+id/Button01"
        android:layout_width="fill_parent"
        android:layout height="wrap content">
    </Button>                                  <!--Button-->
</LinearLayout>
```

代码说明：

- ❑ 该配置文件中的 TextView 控件，声明了该控件的 ID、宽度、高度、显示文字、位置，以及是否为单行。
- ❑ 上述配置文件中声明了 Button 按钮的配置，声明了按钮的 ID、大小，以及名称。

该示例的运行效果如图 8.13 和图 8.14 所示。



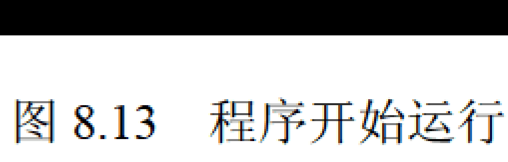

图 8.13　程序开始运行



图 8.14　点击按钮后

第 9 章　Android 传感器

本章将介绍 Android 手机中传感器的基础知识及应用。传感器的应用是 Android 系统的一大亮点，利用各种传感器可以开发出许多有趣的程序。例如，Gameloft 公司发布的都市赛车、EA 公司发布的极品飞车，都是利用手机重力传感器来进行控制的。当然 Android 系统下还有许多其他传感器，本章中将会一一做出介绍。

9.1　传感器简介

本节将对 Android 中的传感器进行简要介绍。传感器对于 Android 系统的手机的迅速崛起非常重要，它可以使得开发者开发出许多意想不到的应用。Android 传感器所包含的功能如表 9.1 所示。

表 9.1　Android 传感器功能

特　　性	属 性 描 述
android.hardware.SensorManager	允许访问 Android 平台传感器的类。并非所有配备 Android 的设备都支持 SensorManager 中的所有传感器，虽然这种可能性让人非常兴奋
android.hardware.SensorListener	在传感器值实时更改时，希望接收更新的类要实现的接口。应用程序实现该接口来监视硬件中一个或多个可用传感器

读者可以从下面的介绍中了解到 Android 系统中包含的传感器的种类。Android 开发包标准有 8 个传感器，如表 9.2 所示。

表 9.2　传感器的种类及名称

类　　型	传感器名称
Sensor.TYPE_ACCELEROMETER	加速度传感器
Sensor.TYPE_GYROSCOPE	陀螺仪传感器
Sensor.TYPE_LIGHT	光照传感器
Sensor.TYPE_MAGNETIC_FIELD	磁场传感器
Sensor.TYPE_ORIENTATION	姿态传感器
Sensor.TYPE_PRESSURE	压力传感器
Sensor.TYPE_PROXIMITY	距离传感器
Sensor.TYPE_TEMPERATURE	温度传感器

- ❑ Sensor.TYPE_ACCELEROMETER：加速度传感器是为了检测物体的加速度的传感器。物体运动加速度也跟着变化，如果能取到加速度，物体受到什么样的作用力

或物体进行什么样的运动，我们就可以知道。使用加速度，我们就能做模拟计步器、物体运动的应用程序。

- ❑ Sensor.TYPE_GYROSCOPE：陀螺仪传感器 values 数组有 3 个元素，values[0]表示绕 X 轴旋转的角速度，values[1]表示绕 Y 轴旋转的角速度，values[2]：表示绕 Z 轴旋转的角速度。
- ❑ Sensor.TYPE_LIGHT：光照传感器主要用来检测手机周围光的强度，与其他传感器不同的是，该传感器只读取一个数值，即手机周围光的强度，且单位为勒克斯（lux）。
- ❑ Sensor.TYPE_MAGNETIC_FIELD：磁场传感器可以感知是磁场的变化，通过磁场传感器可以读出磁场值。利用该传感器可以方便的开发出指南针、罗盘等磁场应用。
- ❑ Sensor.TYPE_ORIENTATION：姿态传感器是使用最多的传感器之一，该传感器主要感应手机方位的变化，捕获的同样是 3 个数，分别代表手机沿 Yaw 轴、Pitch 轴和 Roll 轴转过的角度。
- ❑ Sensor.TYPE_PRESSURE：压力传感器主要用来测量加在手机设备上的压力。
- ❑ Sensor.TYPE_PROXIMITY：距离传感器主要用来测试距离，典型应用为在接听电话时，可以根据光照、声音等条件估计手机与人之间的距离。
- ❑ Sensor.TYPE_TEMPERATURE：温度传感器也是应用较多的传感器，通过运用温度传感器便可开发出手机温度计等有趣的应用。温度传感器根据具体实现手机型号不同，硬件实现有所区别。

下面将讲解传感器的采样率，Android 系统 SensorManager 中采样率有 4 种选择，如表 9.3 所示。

表 9.3　传感器采样率

属 性 名 称	属 性 描 述
SensorManager.SENSOR_DELAY_FASTEST	最快
SensorManager.SENSOR_DELAY_GAME	游戏
SensorManager.SENSOR_DELAY_NORMAL	普通
SensorManager.SENSOR_DELAY_UI	用户界面

说明：上述表格中提到的采样率分别适用于开发不同类型的传感器程序，读者在进行实际开发中要选择合适的采样率。

从本节中读者可以了解到 Android 系统中包含的传感器以及相关知识，下面将会对其进行一一介绍，使读者了解各个传感器的简单开发步骤。

9.2　加速度传感器

本节将介绍加速度传感器的简单应用，下面的例子实现了从手机中读出手机沿 x 轴、y 轴、z 轴 3 个方向的加速度分量，以及其他加速度传感器的相关信息。本章中的测试机型为摩托罗拉 Milestone，此款手机比较具有代表性。

首先介绍一下手机主界面的布局文件 main.xml，代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <TextView
       android:text="加速度传感器的应用"
       android:layout width="fill parent"
       android:layout height="wrap content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
    <TextView
       android:id="@+id/tvX"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
    <TextView
       android:id="@+id/tvY"
       android:layout width="fill parent"
       android:layout height="wrap content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
    <TextView
       android:id="@+id/tvZ"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
        <TextView
       android:id="@+id/info"
       android:layout width="fill parent"
       android:layout height="wrap content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
</LinearLayout>
```

说明： 上述代码中声明了用于显示加速度沿 x 轴、y 轴、z 轴上的各个分量的 TextView 的引用，同时还设置了一个 TextView 的引用，用来显示手机中加速度传感器的各种信息。

下面介绍该程序的主界面 MyAccelerometer_Activity 的实现，代码如下。

```
package com.accelerometer;
import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.widget.TextView;
public class MyAccelerometer_Activity extends Activity
{
    SensorManager mySensorManager;           //SensorManager 对象引用
    Sensor myAccelerometer;                  //加速度传感器
    TextView tvX;                            //TextView 对象引用
    TextView tvY;                            //TextView 对象引用
    TextView tvZ;                            //TextView 对象引用
    TextView info;
   @Override
```

```
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tvX = (TextView)findViewById(R.id.tvX);     //用于显示 x 轴方向加速度
        tvY = (TextView)findViewById(R.id.tvY);     //用于显示 y 轴方向加速度
        tvZ = (TextView)findViewById(R.id.tvZ);     //用于显示 z 轴方向加速度
        info= (TextView)findViewById(R.id.info);
                                    //用于显示手机中加速度传感器的相关信息
        mySensorManager = (SensorManager)
         getSystemService(SENSOR_SERVICE);          //获得 SensorManager 对象
        myAccelerometer=mySensorManager.getDefaultSensor(Sensor.
        TYPE_ACCELEROMETER);
        String str="\n 名字: "+myAccelerometer.getName()+"\n 电池 :"+myAcce-
        lerometer.getPower()+"\n 类型 :"+myAccelerometer.getType()+"\nVendor:
         "+myAccelerometer.getVendor()+
                                    //手机中加速度传感器的相关信息字符串
        "\n 版本: "+myAccelerometer.getVersion()+"\n 幅度: "+myAccelerometer.
        getMaximumRange();
        info.setText(str);                  //将信息字符串赋予名为 info 的 TextView
    }
    @Override
     protected void onResume()              //重写 onResume()方法
     {
         super.onResume();
         mySensorManager.registerListener(mySensorListener,
          myAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
     }
     @Override
     protected void onPause()               //重写 onPause()方法
     {
         super.onPause();
         mySensorManager.unregisterListener(mySensorListener);
                                    //取消注册监听器
     }
     private SensorEventListener mySensorListener = new SensorEventListener()
     {                          //开发实现了 SensorEventListener 接口的传感器监听器
         @Override
         public void onAccuracyChanged(Sensor sensor, int accuracy)
         {
         }
         @Override
         public void onSensorChanged(SensorEvent event)
         {
             float []values=event.values;       //获取 3 个轴方向感上的加速度值
             tvX.setText("x 轴方向上的加速度为: "+values[0]);
             tvY.setText("y 轴方向上的加速度为: "+values[1]);
             tvZ.setText("z 轴方向上的加速度为: "+values[2]);
         }
     };
}
```

代码说明：

- 程序首先声明了需要用到的用于显示沿 x 轴、y 轴、z 轴方向的加速度分量的 TextView 的引用、传感器管理器，以及传感器的引用，并在 onCreate()方法中对其进行初始化。

- 在 onCreate()方法中创建了一个字符串对象，将传感器的相关信息赋予此字符串，并让其显示在名为 info 的 TextView 中。
- 代码中还需实现 SensorEventListener 接口中的 onAccuracyChanged() 和 onSensorChanged()两个方法。其中，onAccuracy()方法在传感器精度发生变化时调用，本程序中没有用到，故其是空实现；onSensorChanged()方法中获取了沿 3 个轴上的加速度分量值，并分别赋予对应的 TextView 引用。
- 在 onResume()方法中为传感器管理器注册监听事件，3 个参数分别为监听器对象、传感器对象和传感器管理器的延迟速率。同时在 onPause()方法中对其取消监听。

以上程序的运行效果界面如图 9.1、图 9.2 和图 9.3 所示。



图 9.1　加速度传感器 1



图 9.2　加速度传感器 2



图 9.3　加速度传感器 3

上面 3 幅图中分别为不同运动状态下的界面。观察可知 x 轴、y 轴、z 轴方向上的加速度分量是在不停的发生变化的。下面显示的是加速度传感器的名称、类型 、版本等加速度传感器信息。

9.3　光照传感器

本节将介绍光照传感器的简单应用，光照传感器也是一种十分有用的传感器，通过光照传感器手机可以自动调节手机屏幕的亮度，还可以写出许多新奇的应用。下面的示例中将介绍如何读出光照传感器的光照强度值，其单位为勒克斯（lux）。

首先介绍主界面布局的 main.xml，代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

```
    >
   <TextView
        android:text="光照传感器的应用"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="24dip"/><!--添加一个 TextView 控件 -->
    <TextView
        android:id="@+id/tvX"
        android:layout width="fill parent"
        android:layout height="wrap content"
        android:textSize="24dip"/><!--添加一个 TextView 控件 -->
        <TextView
        android:id="@+id/info"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="24dip"/><!--添加一个 TextView 控件 -->
</LinearLayout>
```

说明：布局文件中声明了用于显示光强的 TextView，以及用于显示光照传感器的 TextView。

下面介绍程序主界面 MyLightSensor_Activity 的实现，具体代码如下。

```
package com.light;
import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.widget.TextView;
public class MyLightSensor_Activity extends Activity
{
    SensorManager mySensorManager;              //SensorManager 对象引用
    Sensor myLightSensor;                       //光照传感器
    TextView tvX;                               //TextView 对象引用
    TextView info;
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);
       tvX = (TextView)findViewById(R.id.tvX);
       info= (TextView)findViewById(R.id.info);
       //获得 SensorManager 对象
       mySensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
       myLightSensor=mySensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
       String str="\n 名字: "+myLightSensor.getName()+"\n 电池 :"+myLight-
       Sensor.getPower()+"\n 类型 :"+myLightSensor.getType()+"\nVendor:
      "+myLightSensor.getVendor()+"\n 版本: "+myLightSensor. getVersion()
      +"\n 幅度: "+myLightSensor.getMaximumRange();
```

```
        info.setText(str);              //将信息字符串赋予名为 info 的 TextView
    }
    @Override
    protected void onResume()           //重写 onResume()方法
    {
        mySensorManager.registerListener(mySensorListener,
                myLightSensor, SensorManager.SENSOR_DELAY_NORMAL);
        super.onResume();
    }
    @Override
    protected void onPause()                    //重写 onPause()方法
    {
        mySensorManager.unregisterListener(mySensorListener);
                                                //取消注册监听器
        super.onPause();
    }
    //开发实现了 SensorEventListener 接口的传感器监听器
    private SensorEventListener mySensorListener = new SensorEventListener()
    {
        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy)
        {
        }
        @Override
        public void onSensorChanged(SensorEvent event)
        {
            float []values=event.values;
            tvX.setText("光强为:"+values[0]);
        }
    };
}
```

代码说明：

- 代码中首先实现了传感器管理器、传感器、用于显示光强，以及用于显示光照传感器的 TextView 的成员变量的引用。
- 代码中还需实现 SensorEventListener 接口中的 onAccuracyChanged()和 onSensorChanged()两个方法。其中，onAccuracy()方法在传感器精度发生变化时调用，本程序中没有用到，故其是空实现；onSensorChanged()方法中获取了光照强度值，并赋予其对应的 TextView 引用。
- 在 onResume()方法中为传感器管理器注册监听事件，3 个参数分别为监听器对象、传感器对象和传感器管理器的延迟速率。同时，在 onPause()方法中取消对 mySensorManager 传感器管理器的监听。

运行本程序，程序界面如图 9.4、图 9.5 和图 9.6 所示。



图 9.4　光照传感器界面 1



图 9.5　光照传感器界面 2



图 9.6　光照传感器界面 3

以上 3 幅图分别是在不同光照强度下的界面。可以看出，在不同的光照条件下显示的光照强度是不同的。下方的字符串显示的是 Milestone 中自带的光照传感器的名称、类型、幅度等相关信息。读者如果想查看其他光照传感器信息可以参照 API 打印其他信息进行查看。

9.4　温度传感器

本节将介绍温度传感器的简单应用，通过温度传感器可以做出如手机温度计等小应用。本节将简要介绍如何通过温度传感器读出当前手机温度值。

首先介绍一下程序主界面布局文件 main.xml，代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
    <TextView
       android:text="温度传感器的应用"
       android:layout width="fill parent"
       android:layout height="wrap content"
        android:textSize="24dip"
    /><!--添加一个 TextView 控件 -->
    <TextView
       android:id="@+id/tvX"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
        android:textSize="24dip"
    /><!--添加一个 TextView 控件 -->
    <TextView
```

```
        android:id="@+id/info"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="24dip"
    /><!--添加一个 TextView 控件 -->
</LinearLayout>
```

说明： 上述布局代码中声明了用于存放温度值的 TextView，以及用于显示温度传感器相关信息的 TextView。

下面介绍温度传感器示例的主界面 MyTemperatureSensor_Activity 的实现，具体代码如下。

```
package com.temperature;
import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.widget.TextView;
public class MyTemperatureSensor_Activity extends Activity
{
    SensorManager mySensorManager;                  //SensorManager 对象引用
    Sensor myTemperatureSensor;                     //温度传感器
    TextView tvX,info;                              //TextView 对象引用
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tvX = (TextView)findViewById(R.id.tvX); //获取 TextView 的引用
        info=(TextView)findViewById(R.id.info);
        //获得 SensorManager 对象
        mySensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
        myTemperatureSensor=mySensorManager.getDefaultSensor(Sensor.
        TYPE_TEMPERATURE);
        String str="\n 名字: "+myLightSensor.getName()+"\n 电池 :"+
        myLightSensor.getPower()+"\n 类型 :"+myLightSensor.getType()+"\
        nVendor: "+myLightSensor.getVendor()+"\n 版本: "+myLightSensor.
        getVersion()+"\n 幅度: "+myLightSensor.getMaximumRange();
        info.setText(str);              //将信息字符串赋予名为 info 的 TextView
    }
    @Override
    protected void onResume()           //重写 onResume()方法
    {
      mySensorManager.registerListener(mySensorListener,
          myTemperatureSensor,SensorManager.SENSOR_DELAY_NORMAL);
          super.onResume();
    }
    @Override
    protected void onPause()            //重写 onPause()方法
    {
    mySensorManager.unregisterListener(mySensorListener); //取消注册监听器
          super.onPause();
    }
```

```
    //开发实现了 SensorEventListener 接口的传感器监听器
    private SensorEventListener mySensorListener = new SensorEventListener()
    {
        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy)
        {
        }
        @Override
        public void onSensorChanged(SensorEvent event)
        {
            float []values=event.values;
            tvX.setText("温度为:"+values[0]);
        }
    };
}
```

代码说明：

- 代码中首先实现了传感器管理器、传感器、用于显示光强，以及用于显示温度传感器的 TextView 的成员变量的引用。
- 代码中还需实现 SensorEventListener 接口中的 onAccuracyChanged() 和 onSensorChanged()两个方法。其中，onAccuracy()方法在传感器精度发生变化时调用，本程序中没有用到，因此是空实现；onSensorChanged()方法中获取了光照强度值，并赋予其对应的 TextView 引用。
- 在 onResume()方法中为传感器管理器注册监听事件，3 个参数分别为传感器监听器对象、传感器对象和传感器管理器的延迟速率。同时，在 onPause()方法中取消对 mySensorManager 传感器管理器的监听。

程序运行结果如图 9.7、图 9.8 和图 9.9 所示。

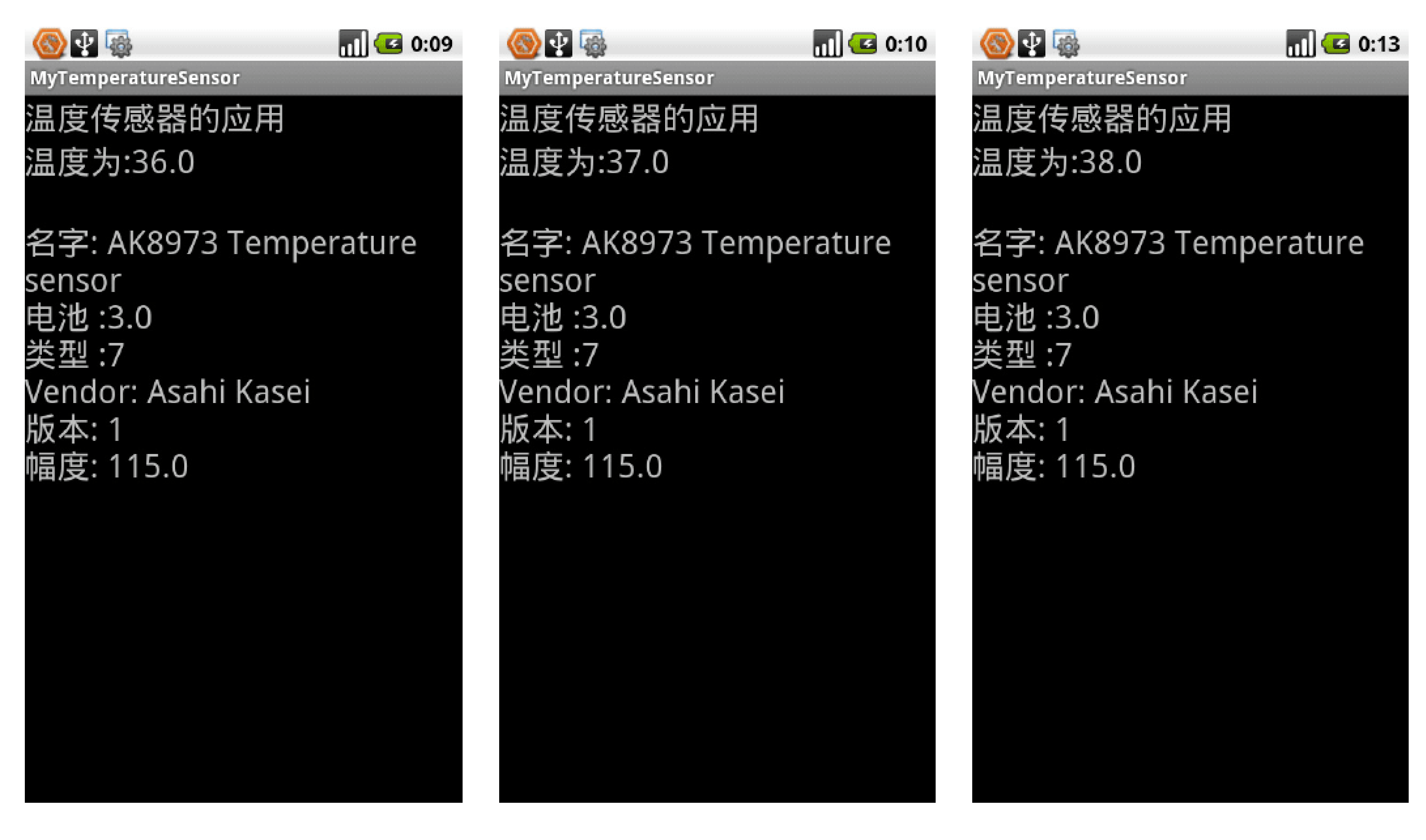


图 9.7　温度传感器 1　　图 9.8　温度传感器 2　　图 9.9　温度传感器 3

上面 3 幅图中分别显示的是手机在不同环境下的温度，可以看出从温度传感器读出的

温度值是随环境的改变而改变的。下面显示的是 Milestone 自带的温度传感器的名称、类型幅度等相关信息。

9.5　磁场传感器

本节将介绍磁场传感器，利用手机磁场传感器我们可以检测出当前位置的磁场，利用磁场传感器可以做出如电子罗盘等许多新奇的应用，本节简要介绍如何读出当前的磁场强度。

首先介绍程序的布局文件 main.xml。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout width="fill parent"
   android:layout height="fill parent">
   <TextView
       android:text="磁场传感器的应用"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
    <TextView
       android:id="@+id/tvX"
       android:layout width="fill parent"
       android:layout height="wrap content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
    <TextView
       android:id="@+id/tvY"
       android:layout_width="fill_parent"
       android:layout height="wrap content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
    <TextView
       android:id="@+id/tvZ"
       android:layout width="fill parent"
       android:layout_height="wrap_content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
        <TextView
       android:id="@+id/info"
       android:layout width="fill parent"
       android:layout height="wrap content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
</LinearLayout>
```

说明：布局文件中声明了用来读取磁场沿 x 轴、y 轴、z 轴的磁场强度的分量的 TextView，以及用来描述磁场传感器的相关信息的 TextView。

下面介绍程序主界面文件 MyMagneticFieldSensor_Activity 的具体实现，代码如下。

```
package com.magnetic_field;
import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
```

```
import android.os.Bundle;
import android.widget.TextView;
public class MyMagneticFieldSensor Activity extends Activity
{
    SensorManager mySensorManager;              //SensorManager 对象引用
    Sensor myMagnetic_field_Sensor;             //磁场传感器
    TextView tvX,tvY,tvZ,info;  //TextView 对象引用
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      tvX = (TextView)findViewById(R.id.tvX);
      tvY = (TextView)findViewById(R.id.tvY);
      tvZ = (TextView)findViewById(R.id.tvZ);
      info=(TextView)findViewById(R.id.info);
      //获得 SensorManager 对象
      mySensorManager = (SensorManager)getSystemService(SENSOR SERVICE);

      myMagnetic field Sensor=mySensorManager.
               getDefaultSensor(Sensor.TYPE MAGNETIC FIELD);
      String str="\n 名字: "+myMagnetic_field_Sensor.getName()+
              "\n 电池 :"+myMagnetic field Sensor.getPower()+
              "\n 类型 :"+myMagnetic field Sensor.getType()+
              "\nVendor: "+myMagnetic field Sensor.getVendor()+
              "\n 版本: "+myMagnetic field Sensor.getVersion()+
              "\n 幅度: "+myMagnetic field Sensor.getMaximumRange();
      info.setText(str);          //将信息字符串赋予名为 info 的 TextView
   }
   @Override
    protected void onResume()       //重写 onResume()方法
    {
       mySensorManager.registerListener(mySensorListener,
              myMagnetic field Sensor, SensorManager.SENSOR DELAY
              NORMAL);
       super.onResume();
    }
    @Override
    protected void onPause()
    {                             //重写 onPause()方法
       mySensorManager.unregisterListener(mySensorListener);
                                  //取消注册监听器
       super.onPause();
    }
    //开发实现了 SensorEventListener 接口的传感器监听器
    private SensorEventListener mySensorListener = new SensorEventListener()
    {
       @Override
       public void onAccuracyChanged(Sensor sensor, int accuracy)
       {
       }
       @Override
```

```
        public void onSensorChanged(SensorEvent event)
        {
            float []values=event.values;
            System.out.println(values.length+"=======");
            tvX.setText("X 轴方向磁场为:"+values[0]);
                                        //为各个 TextView 的引用赋值
            tvY.setText("Y 轴方向磁场为:"+values[1]);
            tvZ.setText("Z 轴方向磁场为:"+values[2]);
        }
    };
}
```

代码说明：

- 程序首先声明了用于显示沿 x 轴、y 轴、z 轴方向的磁场强度分量的 TextView 的引用、传感器管理器，以及传感器的引用，并在 onCreate()方法中对其进行初始化。
- 在 onCreate()方法中创建了一个字符串对象，将磁场传感器的相关信息赋予此字符串，并让其显示在名为 info 的 TextView 中，便于观察磁场传感器的相关信息。
- 代码中还需实现 SensorEventListener 接口中的 onAccuracyChanged() 和 onSensorChanged()两个方法。其中，onAccuracy()方法在传感器精度发生变化时调用，本程序中没有用到，故其是空实现；onSensorChanged()方法中获取了沿 3 个轴上的磁场强度分量值，并分别赋予对应的 TextView 引用。其中磁场强度的单位为微特斯拉（uT）。
- 在 onResume()方法中为传感器管理器注册监听事件，3 个参数分别为监听器对象，传感器对象和传感器管理器的延迟速率。同时在 onPause()方法中对其取消监听。

运行本程序，结果如图 9.10、图 9.11 和图 9.12 所示。



图 9.10　磁场传感器 1　　图 9.11　磁场传感器 2　　图 9.12　磁场传感器 3

由上面 3 幅图中可以看出，手机所处的当前环境中的磁场值是不固定的，并且是不断

变化的，其可能会受到某些电磁信号的干扰。

下面是 MileStone 自带的磁场传感器包括名称，类型、版本在内的相关信息。读者若想要了解传感器的其他信息，可参照 API 打印其他信息以进行查看。

9.6　姿态传感器

姿态传感器又称为方向传感器，姿态传感器主要感应手机方位的变化，每次读取的是静态的状态值。本节将简要介绍如何读出分别沿 3 个轴转过的角度。

首先是布局文件 main.xml，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout width="fill parent"
   android:layout height="fill parent">
   <TextView
       android:text="姿态传感器的应用"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
    <TextView
       android:id="@+id/tvX"
       android:layout width="fill parent"
       android:layout height="wrap content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
    <TextView
       android:id="@+id/tvY"
       android:layout_width="fill_parent"
       android:layout height="wrap content"
       android:textSize="24dip" /><!--添加一个 TextView 控件 -->
    <TextView
       android:id="@+id/tvZ"
       android:layout width="fill parent"
       android:layout_height="wrap_content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
        <TextView
       android:id="@+id/info"
       android:layout width="fill parent"
       android:layout height="wrap content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
</LinearLayout>
```

说明： 布局文件中声明了用来读取手机分别沿 Yaw 轴、Pitch 轴、Roll 轴转动的角度 TextView，以及用来描述姿态传感器的相关信息的 TextView。

```
package com.orientation;
import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
```

```
import android.widget.TextView;
public class MyOrientation Activity extends Activity {
     SensorManager mySensorManager;//SensorManager 对象引用
     Sensor myOrientation_Sensor;//方向传感器
     TextView tvX,tvY,tvZ,info; //TextView 对象引用
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tvX = (TextView)findViewById(R.id.tvX);
        tvY = (TextView)findViewById(R.id.tvY);
        tvZ = (TextView)findViewById(R.id.tvZ);
        info=(TextView)findViewById(R.id.info);
        //获得 SensorManager 对象
        mySensorManager = (SensorManager)getSystemService(SENSOR SERVICE);
        myOrientation Sensor=mySensorManager.getDefaultSensor(Sensor.
        TYPE ORIENTATION);
        String str="\n 名字: "+myOrientation Sensor.getName()+"\n 电池 :
        "+myOrientation_Sensor.getPower()+"\n 类型 :"+myOrientation_Sensor.
        getType()+"\nVendor: "+myOrientation_Sensor.getVendor()+"\n 版本:
        "+myOrientation_Sensor.getVersion()+"\n 幅度: "+myOrientation_
        Sensor.getMaximumRange();
        info.setText(str);            //将信息字符串赋予名为 info 的 TextView
    }
    @Override
     protected void onResume()        //重写 onResume()方法
     {
         mySensorManager.registerListener(mySensorListener,
                 myOrientation_Sensor, SensorManager.SENSOR_DELAY_NORMAL);
         super.onResume();
     }
     @Override
     protected void onPause()         //重写 onPause()方法
     {
         mySensorManager.unregisterListener(mySensorListener);
                                      //取消注册监听器
         super.onPause();
     }
     //开发实现了 SensorEventListener 接口的传感器监听器
     private SensorEventListener mySensorListener = new SensorEventListener()
     {
         @Override
         public void onAccuracyChanged(Sensor sensor, int accuracy)
         {
         }
         @Override
         public void onSensorChanged(SensorEvent event)
         {
             float []values=event.values;
             System.out.println(values.length+"=======");
             tvX.setText("手机沿 Yaw 轴转过的角度为:"+values[0]);
             tvY.setText("手机沿 Pitch 轴转过的角度为:"+values[1]);
             tvZ.setText("手机沿 Roll 轴转过的角度为:"+values[2]);
         }
     };
}
```

代码说明：

- 程序首先声明了用于显示沿 Yaw 轴、Pitch 轴、Roll 轴方向转过的角度的 TextView 的引用、传感器管理器，以及传感器的引用，并在 onCreate()方法中对其进行初始化。
- 在 onCreate()方法中创建了一个字符串对象，将姿态传感器的相关信息赋予此字符串，并让其显示在名为 info 的 TextView 中，便于观察姿态传感器的相关信息。
- 代码中还需实现 SensorEventListener 接口中的 onAccuracyChanged()和 onSensorChanged()两个方法。其中，onAccuracy()方法在传感器精度发生变化时调用，本程序中没有用到，故其是空实现；onSensorChanged()方法中获取了沿 3 个轴转过的角度值，并分别赋予对应的 TextView 引用。
- 读出的 3 个值的单位为均为度（degree）。
- 在 onResume()方法中为传感器管理器注册监听事件，3 个参数分别为监听器对象、传感器对象和传感器管理器的延迟速率。同时在 onPause()方法中对其取消监听。

运行本程序，结果如图 9.13、图 9.14 和图 9.15 所示。



图 9.13　方向传感器 1　　图 9.14　方向传感器 2　　图 9.15　方向传感器 3

上面 3 幅图分别是在手机在不同姿态下的截图，可见在不同姿态下手机沿 Yaw 轴、Pitch 轴、Roll 轴转过的角度是不一样的。

下面显示的是 MileStone 的姿态传感器的名称、类型、版本，以及幅度等相关信息。

9.7　距离传感器

本节将介绍距离传感器。距离传感器主要用于感应周围靠近的物体，用过 Android 系统手机的读者可能对其比较熟悉，当打电话时将手机放在耳边，手机屏幕背景灯会自动关闭，当手机离开耳边时，手机屏幕背景灯会自动打开。这是一个很好的距离传感器的应用。

下面介如何获取距离的值。

首先介绍一下布局文件 main.xml 的具体实现，代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <TextView
       android:text="距离传感器的应用"
       android:layout width="fill parent"
       android:layout height="wrap content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
    <TextView
       android:id="@+id/tvX"
       android:layout width="fill parent"
       android:layout height="wrap content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
       <TextView
       android:id="@+id/info"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
</LinearLayout>
```

说明： 布局文件中声明了用于存放距离值的 TextView，以及用于存放手机中距离传感器的相关信息的 TextView。

下面介绍程序主界面的具体实现，代码如下。

```
package com.proximity;
import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.widget.TextView;
public class MyProximity_Activity extends Activity {
    SensorManager mySensorManager;//SensorManager 对象引用
    Sensor myProximity_Sensor;//距离传感器
    TextView tvX,info;  //TextView 对象引用
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      tvX = (TextView)findViewById(R.id.tvX); //获取 TextView 的引用
      info=(TextView)findViewById(R.id.info);
      //获得 SensorManager 对象
      mySensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);

      myProximity_Sensor=mySensorManager.getDefaultSensor(Sensor.TYPE_
      PROXIMITY);
      String str="\n 名字: "+myProximity_Sensor.getName()+"\n 电池 :
      "+myProximity_Sensor.getPower()+"\n 类型 :"+myProximity_Sensor.
      getType()+"\nVendor: "+myProximity_Sensor.getVendor()+"\n 版本:
      "+myProximity_Sensor.getVersion()+"\n 幅度: "+myProximity_Sensor.
```

```
        getMaximumRange();
        info.setText(str);              //将信息字符串赋予名为 info 的 TextView
    }
    @Override
    protected void onResume()           //重写 onResume()方法
    {
        mySensorManager.registerListener(mySensorListener,
                myProximity Sensor, SensorManager.SENSOR DELAY NORMAL);
        super.onResume();
    }
    @Override
    protected void onPause()            //重写 onPause()方法

    {
        mySensorManager.unregisterListener(mySensorListener);
                                        //取消注册监听器
        super.onPause();
    }
    //开发实现了 SensorEventListener 接口的传感器监听器
    private SensorEventListener mySensorListener = new SensorEventListener()
    {
        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy)
        {
        }
        @Override
        public void onSensorChanged(SensorEvent event)
        {
            float []values=event.values;
            tvX.setText("手机距离物体的距离为:"+values[0]);
        }
    };
}
```

代码说明：

- 程序首先声明了用于显示距离 TextView 的引用、传感器管理器，以及传感器的引用，并在 onCreate()方法中对其进行初始化。
- 在 onCreate()方法中创建了一个字符串对象，将距离传感器的相关信息赋予此字符串，并让其显示在名为 info 的 TextView 中，便于观察距离传感器的相关信息。
- 代码中还需实现 SensorEventListener 接口中的 onAccuracyChanged()和 onSensor-Changed()两个方法。其中，onAccuracy()方法在传感器精度发生变化时调用，本程序中没有用到，故其是空实现；onSensorChanged()方法中获取了距离值，并赋予其对应的 TextView 引用。
- 距离传感器在注册监听后仅捕获一个参数 values[0]，该参数代表靠近传感器的物体距离，以厘米为单位。
- 在 onResume()方法中为传感器管理器注册监听事件，3 个参数分别为监听器对象，传感器对象和传感器管理器的延迟速率。同时在 onPause()方法中对其取消监听。

运行程序，界面如图 9.16、图 9.17 和图 9.18 所示。

图 9.16 是用手贴近手机时，可以看到界面中所显示的距离为 0；图 9.17 是手离开手机时手机显示的距离，当再次用手接近手机时，距离重新变为 0，如图 9.18 所示。



图 9.16　距离传感器 1　　图 9.17　距离传感器 2　　图 9.18　距离传感器 3

下面显示的是 MileStone 的距离传感器的名称、类型、版本，以及幅度等相关信息。

9.8　陀螺仪传感器

本节将介绍陀螺仪的简单应用，手机逆时针旋转时，角速度为正值，顺时针旋转时，角速度为负值。陀螺仪传感器经常被用来计算手机已转动的角度。

首先介绍一下布局文件 main.xml，代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout width="fill parent"
   android:layout height="fill parent">
   <TextView
       android:text="陀螺仪传感器的应用"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
    <TextView
       android:id="@+id/tvX"
       android:layout width="fill parent"
       android:layout height="wrap content"
       android:textSize="24dip"/><!--添加一个 TextView 控件 -->
    <TextView
       android:id="@+id/tvY"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:textSize="24dip" /><!--添加一个 TextView 控件 -->
    <TextView
       android:id="@+id/tvZ"
       android:layout_width="fill_parent"
```

```
        android:layout height="wrap content"
        android:textSize="24dip"/><!--添加一个 TextView 控件 -->
</LinearLayout>
```

🔔**说明：**布局文件中声明了用于显示手机沿 x、y、z 轴旋转的角速度的 TextView。

下面介绍程序主界面 MyGyroscope_Activity 的具体实现，代码如下。

```
package com.yroscope;
import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.widget.TextView;
public class MyGyroscope Activity extends Activity {
    SensorManager mySensorManager;//SensorManager 对象引用
    Sensor myGyroscope;                  //陀螺仪传感器
    TextView tvX;                        //TextView 对象引用
    TextView tvY;                        //TextView 对象引用
    TextView tvZ;                        //TextView 对象引用
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tvX = (TextView)findViewById(R.id.tvX);      //用于显示 x 轴方向加速度
        tvY = (TextView)findViewById(R.id.tvY);      //用于显示 y 轴方向加速度
        tvZ = (TextView)findViewById(R.id.tvZ);      //用于显示 z 轴方向加速度
        mySensorManager = (SensorManager)getSystemService(SENSOR SERVICE);
                                         //获得 SensorManager 对象
        myGyroscope=mySensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
    }
    @Override
    protected void onResume()            //重写 onResume()方法
    {
        super.onResume();
        mySensorManager.registerListener(mySensorListener,
                myGyroscope, SensorManager.SENSOR DELAY NORMAL);
    }
    @Override
    protected void onPause()             //重写 onPause()方法
    {
        super.onPause();
        mySensorManager.unregisterListener(mySensorListener);
                                         //取消注册监听器
    }
    private SensorEventListener mySensorListener = new SensorEventListener()
    {                    //开发实现了 SensorEventListener 接口的传感器监听器
        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy)
        {
        }
        @Override
        public void onSensorChanged(SensorEvent event)
        {
            float []values=event.values;        //获取 3 个轴方向感上的加速度值
            System.out.println(values.length);
```

```
            tvX.setText("沿 x 轴旋转的角速度为："+values[0]);
            tvY.setText("沿 y 轴旋转的角速度为："+values[1]);
            tvZ.setText("沿 z 轴旋转的角速度为："+values[2]);
        }
    };
}
```

代码说明：

- 程序首先声明了用于显示距离 TextView 的引用、传感器管理器，以及传感器的引用，并在 onCreate()方法中对其进行初始化。
- 代码中还需实现 SensorEventListener 接口中的 onAccuracyChanged() 和 onSensorChanged()两个方法。其中 onAccuracy()方法在传感器精度发生变化时调用，本程序中没有用到，故其是空实现；onSensorChanged()方法中获取了距离值，并赋予其对应的 TextView 引用。
- 陀螺仪传感器在注册监听后仅捕获 3 个参数 values[0]、values[1]和 values[2]，这些参数代表分别沿 x、y、z 轴旋转的角速度。
- 在 onResume()方法中为传感器管理器注册监听事件，3 个参数分别为监听器对象、传感器对象和传感器管理器的延迟速率。同时在 onPause()方法中对其取消监听。

运行本程序，结果如图 9.19、图 9.20 和图 9.21 所示。



图 9.19　陀螺仪传感器 1　　图 9.20　陀螺仪传感器 2　　图 9.21　陀螺仪传感器 3

说明：上面 3 幅图是在旋转手机时的截图，可见在旋转手机的过程中沿 x 轴、y 轴、z 轴旋转过的角速度是在不断变化的。

第 10 章　Android 游戏开发基础

游戏开发一直是各个平台上不可或缺的一部分，也是软件开发中最令人感兴趣的部分之一，Android 平台也不例外。

如果说前面章节介绍的基本控件的使用是在一个框架里搭积木的话，游戏开发的框架就像是用画笔在画布上作画。游戏开发涉及的范围很广，内容十分丰富，本章只对开发所需的基本元素及其操作方法做一些介绍。

10.1　View 框架

既然是要绘画，就要准备好一个架子，铺上画布，然后用画笔作画。在 Android 平台中，view 框架是最基础也是最常用的架子。通过下面的小例子，可以看到如何最简单的使用 view 框架。流程非常简单：准备一个继承了 view 的子类，在上面写一段文字，然后在 Activity 中调用。首先是自定义的 view 文件，代码如下：

```
public class MyView extends View{
    public MyView(Context context) {
        super(context);                         //调用父类构造函数
    }
    public void onDraw(Canvas canvas){          //自动调用描绘方法
       Paint mPaint = new Paint();              //实例化 Paint
       mPaint.setColor(Color.RED);              //定义 Paint 对象颜色
       mPaint.setTextSize(28);                  //定义 Paint 对象文字大小
       mPaint.setAntiAlias(true);               //开启文字抗锯齿
       canvas.drawRGB(255, 255, 255);           //Canvas 对象描绘背景色
       canvas.drawText("Hello World!", 20, 120, mPaint);   //描绘文字
    }
}
```

代码说明：

- 要使用 view 框架，需要自定义一个类来继承 android.view.View 类，并且调用父类的构造函数。
- 自定义的类要重写父类的 onDraw()方法，将描绘内容的代码写在里面。
- onDraw()方法带有一个 android.graphics.Canvas 类的对象做参数。
- 可以将 Canvas 对象看作是描绘图画的画布，在画布上面描绘需要的内容，在描绘内容时，可以通过 android.graphics.Paint 类的对象设置参数，Paint 类就是画笔。
- 在本例中，Paint 类的对象设置了 3 个属性，分别是字体颜色、字体大小及消除字体锯齿。

- Canvas 类的 drawText 方法用来描绘文字，该方法有多种重载方式。本例中使用的方式需要 4 个参数，第 1 个参数为 String 类型，即文字内容；第 2 个参数为 float 类型，即居容器左上角的 x 距离；第 3 个参数为 float 类型，即居容器左上角的 y 距离；第 4 个参数为 Paint 类型，即定义的 Paint 类对象。

准备好了自定义的 view 类，接下来就是在 Activity 中调用，MyActivity.java 代码如下：

```
public class MyActivity extends Activity{
    private MyView mGameView = null;        //定义 MyView 对象
    public void onCreate(Bundle savedInstanceState){
       super.onCreate(savedInstanceState);
       this.mGameView = new MyView(this);  //实例化 MyView 对象
       setContentView(mGameView);           //将 MyView 对象添加进 Activity
    }
}
```

程序运行效果如图 10.1 所示。

上面的例子很简单，简单到没有丝毫“动画”的效果。下面来做一个复杂一点的例子，通过程序来不停地改变文字的颜色。

程序的流程：在 Activity 文件中使用新的线程不断“刷新”view 文件，在 view 文件中通过变量控制选择不同的颜色来描绘文字。改进后的自定义 view 文件代码如下：

图 10.1　view 框架简单使用

```
public class MyView extends View {
    private int count = 0; //定义控制变量
    public MyView(Context context) {
        super(context);     //调用父类构造方法
    }

    public void onDraw(Canvas canvas) {
        int color=0;        //定义颜色变量
        switch (count) {//判断当前控制变量的值
        case 0:
            color=Color.BLACK;          //取黑色值
            break;
        case 1:
            color=Color.BLUE;           //取蓝色值
            break;
        case 2:
            color=Color.GREEN;          //取绿色值
            break;
        case 3:
            color=Color.RED;            //取红色值
            break;
        case 4:
            color=Color.YELLOW;         //取黄色值
            break;
        }
        count=count+1;                  //累加控制变量
        if(count>4){                    //判断控制变量上限
            count=0;                    //初始化控制变量
        }
```

```
        Paint mPaint = new Paint();     //实例化 Paint
        mPaint.setColor(color);         //定义 Paint 对象颜色
        mPaint.setTextSize(28);         //定义 Paint 对象文字大小
        mPaint.setAntiAlias(true);      //开启文字抗锯齿
        canvas.drawRGB(255, 255, 255); // Canvas 对象描绘背景色
        canvas.drawText("Hello World!", 20, 120, mPaint);     //描绘文字
    }
}
```

代码说明如下：

android.graphics.Color 类提供了一些静态常量和方法，用来表示某一种颜色。这些静态常量的值和方法的返回值都是整数类型。

下面是修改后的 Activity 类，代码如下：

```
public class MyActivity extends Activity{
    private MyView mGameView = null;        //定义 MyView 对象
    public void onCreate(Bundle savedInstanceState){
       super.onCreate(savedInstanceState);
       this.mGameView = new MyView(this);  //实例化 MyView 对象
       setContentView(mGameView);          //将 MyView 对象添加进 Activity
       new Thread(new Runnable() {         //新的线程
            public void run() {
                try {
                    while(true){            //死循环
                        Message m = new Message(); //定义 Message 对象
                        viewHandler.sendMessage(m);
                                            //向 Handler 传递 Message 对象
                        Thread.sleep(1000); //停歇 1000 毫秒
                    }
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }).start();                         //线程启动
    }
    Handler viewHandler = new Handler() {  //定义接受信息的 Handler
        public void handleMessage(Message msg) {
            myView.invalidate();            //刷新 MyView
            super.handleMessage(msg);
        }
    };
}
```

代码说明：

- 要更新自定义 view 类上描绘的内容，需要调用该类的 invalidate()方法。这个方法有多种重载形式，本例中使用的方式不带参数，会将当前描绘的所有内容全部废除。
- 调用 invalidate()方法后，系统会自动调用 onDraw()方法重新绘制内容。Invalidate()方法只能在 UI 界面的类的线程中使用，如果要在其他的类的线程中使用，应调用 postInvalidate()方法。
- 本例是一个演示案例，所以线程中的代码写成一个死循环。

程序运行效果如图 10.2 所示。

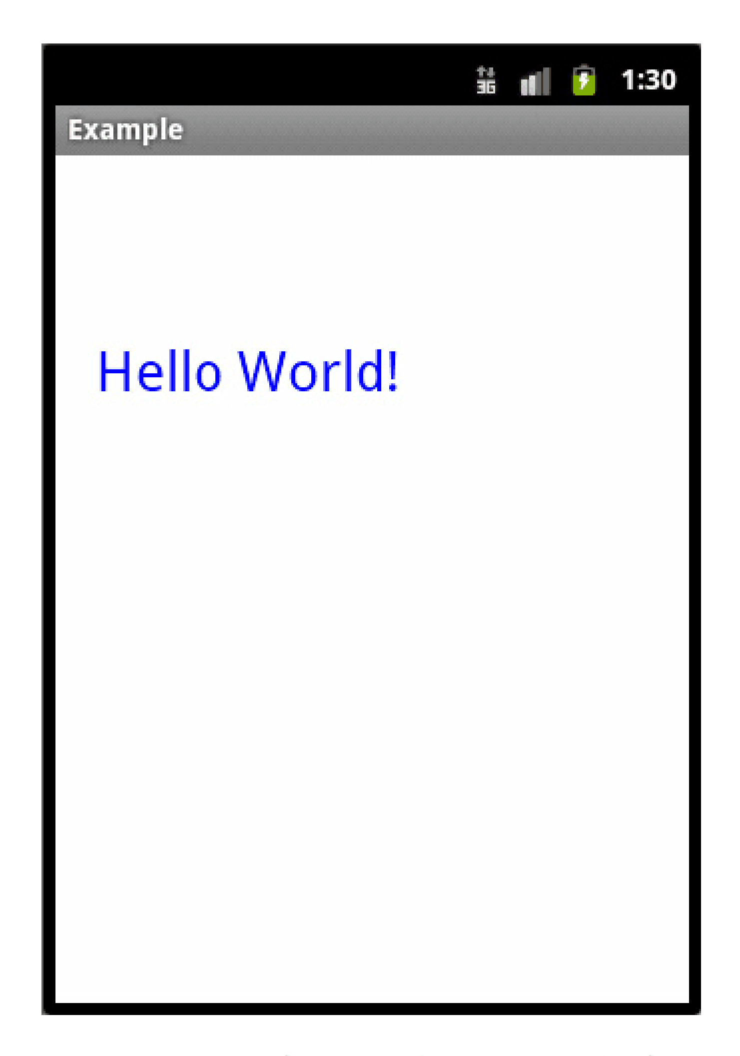


图 10.2　线程更新 view 内容

10.2　SurfaceView 框架

SurfaceView 是 View 的子类，也是经常使用的“架子”之一。在使用上，SurfaceView 与 View 最大的区别是不需要通过 UI 的线程来更新绘图。

下面是一个使用 SurfaceView 通过多张渐变图片实现烟花动画效果的例子，首先是继承了 SurfaceView 的类，代码如下：

```
public class MySurfaceView extends SurfaceView implementsSurfaceHolder.
Callback, Runnable {
    private SurfaceHolder mSurfaceHolder = null;
                                          //定义 SurfaceHolder 对象
    private int count = 0;                //定义计数变量
    private Context cot;                  //定义 Context 对象
    public MySurfaceView(Context context) {
        super(context);                   //调用父类构造方法
        cot = context;                    //为 Context 对象赋值
        mSurfaceHolder = this.getHolder(); //获取 SurfaceHolder 对象实例
        mSurfaceHolder.addCallback(this);  //添加 Callback 接口
    }

    public void surfaceChanged(SurfaceHolder holder, int format, int width,
            int height) {             //当 SurfaceView 实例尺寸改变时调用
    }

    public void surfaceCreated(SurfaceHolder holder) {
                                      //当 SurfaceView 实例创建时调用
        new Thread(this).start();     //启动新的线程
    }

    public void surfaceDestroyed(SurfaceHolder holder) {
                                      //当 SurfaceView 实例销毁时调用
    }

    public void run() {               //重写 run()方法
```

```
        while (true) {                //死循环
            try {
                Thread.sleep(100);  //线程停止 100 毫秒
            } catch (Exception e) {
            }
            synchronized (mSurfaceHolder) {
                                      //同步锁定 mSurfaceHolder 对象
                Draw();               //调用描绘方法
            }
        }
    }

    public void Draw() {
        Canvas canvas = mSurfaceHolder.lockCanvas();
                                //SurfaceHolder 锁定并获得 Canvas 对象
        if (mSurfaceHolder == null || canvas == null) {
            return;
        }
        InputStream iso;              //定义 InputStream 对象
        Bitmap bitmap = null;         //定义 Bitmap 对象
        try {
            iso = cot.getAssets().open("pics/" + count + ".png");
                                      //获取 assets 文件夹下图片
            bitmap = BitmapFactory.decodeStream(iso);
                                      //将 InputStream 转化为 Bitmap
        } catch (IOException e) {
            e.printStackTrace();
        }
        canvas.drawRGB(255, 255, 255);      //绘制背景色
        canvas.drawBitmap(bitmap, 300 / 2 - bitmap.getWidth() / 2,
                                      //绘制图片
                300 / 2 - bitmap.getHeight() / 2, null);
        mSurfaceHolder.unlockCanvasAndPost(canvas); //解锁并显示内容
        count++;                      //累加计数变量
        if (count > 11) {             //判断计数变量的数值
            count = 0;                //初始化计数变量
        }
    }
}
```

代码说明：

- android.view.SurfaceHolder.Callback 接口提供了一组方法，可以在 SurfaceView 发生变化时接收信息。使用 SurfaceView 对象的 addCallback()方法就可以为该对象添加回调。
- 实现 Callback 接口需要重写 3 个方法，分别在 SurfaceView 对象创建、销毁及改变尺寸的时候触发。
- 使用 SurfaceView 对象来绘图，需要先锁定界面，在绘制完后再解锁放开界面。这个操作需要 android.view.SurfaceHolder 对象来完成。
- SurfaceView 的 getHolder()方法或可以返回一个可以控制该 SurfaceView 的 SurfaceHolder 对象。
- 调用 SurfaceHolder 的 lockCanvas()方法会锁定 SurfaceView，并返回一个可以在该 SurfaceView 上进行绘制的 Canvas 对象。
- 调用 SurfaceHolder 的 unlockCanvasAndPost()方法，将 SurfaceView 的锁定解除，

并将绘制的内容显示在屏幕上。unlockCanvasAndPost 的参数为 Canvas 对象。

- Canvas 对象的 drawRGB 可以绘制背景颜色，参数是红、绿、蓝三原色，取值范围从 0～255。另外有 drawARGB()方法，可以绘制带透明的背景，参数是 alpha 及红、绿、蓝三原色，取值范围从 0～255。
- Canvas 类的 drawBitmap()方法用来绘制位图。drawBitmap 方法有多种重载形式，本例中的形式带有 4 个参数，第 1 个参数为 android.graphics.Bitmap 类型，即需要绘制的图像；第 2 个参数为 float 类型，即图像距容器左上角的 x 距离；第 3 个参数为 float 类型，即图像距容器左上角的 y 距离；第 4 个参数为 Paint 类型，即定义的 Paint 类对象。

下面是调用的 Activity 类，代码如下：

```
public class MyActivity extends Activity{
    private MySurfaceView mySurfaceView;
   public void onCreate(Bundle savedInstanceState){
       super.onCreate(savedInstanceState);
       mySurfaceView = new MySurfaceView(this);
       setContentView(mySurfaceView);
   }
}
```

程序运行效果如图 10.3 所示。

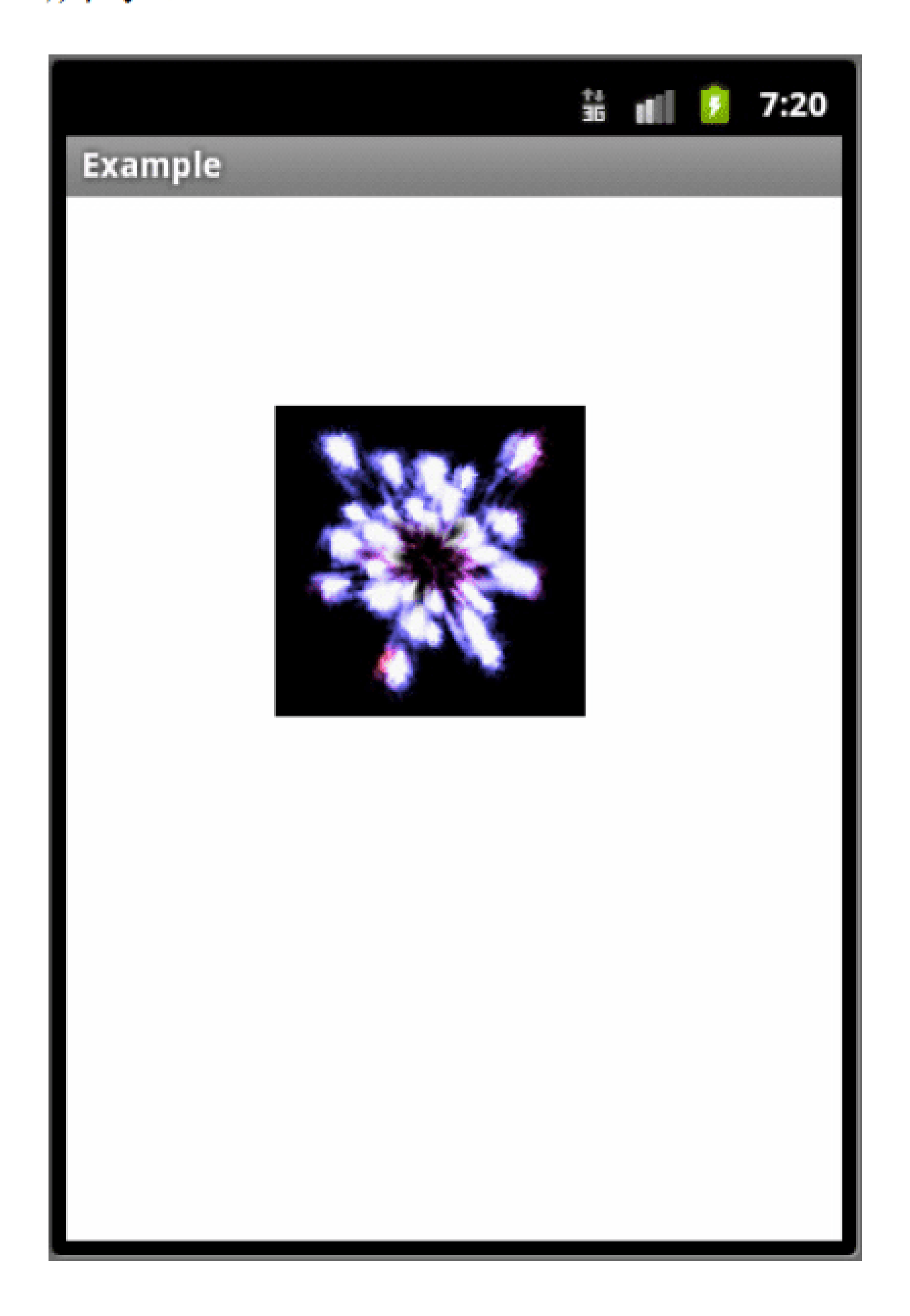


图 10.3　SurfaceView 使用

10.3　Canvas 对象绘制图形

前面一节中，主要研究了两个动画制作的框架。本节中，研究的目标转向 Canvas 对象，将尝试着使用 Canvas 对象来绘制各种几何图形。首先，准备程序案例的界面，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
                                                    <!-- 整体线形布局 -->
    android:orientation="vertical" android:layout_width="fill_parent"
                                                    <!--垂直-->
    android:layout_height="fill_parent" android:background="#ffffffff">
    <LinearLayout android:orientation="horizontal" <!--整体线形布局-->
        android:layout_width="wrap_content" android:layout_height=
        "wrap_content"                               <!--水平-->
        android:layout_marginBottom="15px">
        <Button android:id="@+id/button1" android:layout_width="80px"
                                                    <!--定义按钮-->
            android:layout_height="wrap_content" android:text="图形"
            android:layout_marginTop="15px" />
        <Button android:id="@+id/button2" android:layout_width="80px"
                                           <!--定义按钮-->
            android:layout_height="wrap_content" android:text="填充"
            android:layout_marginTop="15px" />
        <Button android:id="@+id/button3" android:layout_width="80px"
                                           <!--定义按钮-->
            android:layout_height="wrap_content" android:text="渐变"
            android:layout_marginTop="15px" />
    </LinearLayout>
    <com.example.MySurfaceView android:id="@+id/sview" <!--自定义控件-->
        android:layout_gravity="center" android:layout_width="300px"
        android:layout_height="300px" />
</LinearLayout>
```

代码说明：

界面布局中总体是一个垂直线形布局，布局中上半部分是一个水平线形布局，包含 3 个按钮；下半部分是一个我们自定义继承 SurfaceView 的类。

程序的流程很简单，界面上的 3 个按钮，单击后分别描绘空心的几何图形、使用颜色填充的几何图形，以及使用渐变色填充的几何图形。下面，准备一个继承 SurfaceView 类的自定义类，用来描绘各种几何图形，代码如下：

```
public class MySurfaceView extends SurfaceView implementsSurfaceHolder.Callback,
Runnable {
    private SurfaceHolder mSurfaceHolder = null;  //定义 SurfaceHolder 对象
    private int count = -1;                    //定义类别控制变量
    private boolean pan = true;                //定义循环控制变量
    public MySurfaceView(Context context, AttributeSet attrs) {
        super(context, attrs);                 //调用父类构造方法
        mSurfaceHolder = this.getHolder();     //获取 SurfaceHolder 对象实例
        mSurfaceHolder.addCallback(this); //为 SurfaceHolder 对象实例添加回调
    }

    public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int
    arg3) {
    }                                  // SurfaceView 对象实例尺寸改变时触发
    // SurfaceView 对象实例创建时触发
    public void surfaceCreated(SurfaceHolder holder) {
        new Thread(this).start();      //开始新的线程
    }

    public void surfaceDestroyed(SurfaceHolder holder) {
```

```
                                        //SurfaceView 对象实例销毁时触发
        count = -1;                     //初始化类别控制变量
        pan = false;                    //初始化循环控制变量
    }

    public void run() {                 //线程工作方法
        while (pan) {                   //循环条件
            synchronized (mSurfaceHolder) {     //同步 mSurfaceHolder 对象
                Canvas canvas = mSurfaceHolder.lockCanvas();
                                        //锁定并获得 Canvas 类对象
                canvas.drawColor(Color.WHITE);      //绘制背景色
                switch (this.getCount()) {          //判断绘制条件
                case 0:
                    drawGraphic(canvas);            //描绘几何图形
                    break;
                case 1:
                    fillGraphic(canvas);            //描绘颜色填充几何图形
                    break;
                case 2:
                    linearGradientGraphic(canvas); //描绘渐变色填充几何图形
                    break;
                }
                mSurfaceHolder.unlockCanvasAndPost(canvas);
                                                   //解锁并显示描绘内容
            }
            try {
                Thread.sleep(1000);                 //线程休眠 1 000 毫秒
            } catch (Exception e) {
            }
        }
    }

    public void drawGraphic(Canvas canvas) {
        Paint mPaint = new Paint();                 //实例化 Paint 对象
        mPaint.setStyle(Paint.Style.STROKE);        //设置图像为空心
        mPaint.setStrokeWidth(3);                   //设置边框宽度
        mPaint.setAntiAlias(true);                  //设置抗锯齿
        Draw(canvas, mPaint);                       //开始绘图
    }

    public void fillGraphic(Canvas canvas) {
        Paint mPaint = new Paint();                 //实例化 Paint 对象
        mPaint.setStyle(Paint.Style.FILL);          //设置图像为填充
        mPaint.setColor(Color.BLUE);                //设置颜色为蓝色
        mPaint.setAntiAlias(true);                  //设置抗锯齿
        Draw(canvas, mPaint);                       //开始绘图
    }

    public void linearGradientGraphic(Canvas canvas) {
        Shader linearShader = new LinearGradient(0, 0, 25, 25, Color.BLACK,
                Color.WHITE, Shader.TileMode.MIRROR);
                                            //实例化 LinearGradient 对象
        Shader sweepShader = new SweepGradient(60, 130, Color.BLACK,
                Color.WHITE);               //实例化 SweepGradient 对象
        Shader radialShader = new RadialGradient(150, 130, 40, Color.BLACK,
                Color.WHITE, Shader.TileMode.MIRROR);
                                            //实例化 RadialGradient 对象
```

```
        Draw(canvas, linearShader, sweepShader,radialShader);
    }

    public void Draw(Canvas canvas, Paint mPaint) {
        canvas.drawCircle(40, 40, 30, mPaint);          //绘制圆形
        canvas.drawRect(100, 10, 160, 70, mPaint);      //绘制正方形
        canvas.drawRect(10, 90, 100, 150, mPaint);      //绘制长方形
        RectF rf = new RectF(120, 90, 200, 150);        //实例化 RectF 对象
        canvas.drawRoundRect(rf, 10, 10, mPaint);       //绘制圆角矩形
        rf = new RectF(10, 180, 90, 220);               //实例化 RectF 对象
        canvas.drawOval(rf, mPaint);                    //绘制椭圆
        Path path = new Path();                         //实例化 Path
        path.moveTo(110, 180);                          //定义起始点
        path.lineTo(110, 240);                          //连接点
        path.lineTo(170, 240);                          //连接点
        path.close();                                   //闭合连线
        canvas.drawPath(path, mPaint);                  //绘制多边形
    }

    public void Draw(Canvas canvas, Shader shader1, Shader shader2, Shader
    shader3) {
        Paint mPaint = new Paint();                     //实例化 Paint 对象
        mPaint.setAntiAlias(true);                      //设置抗锯齿
        mPaint.setShader(shader1);                      //加载 shader1
        canvas.drawCircle(40, 40, 30, mPaint);          //绘制圆形
        canvas.drawRect(100, 10, 160, 70, mPaint);      //绘制正方形
        mPaint.setShader(shader2);                      //加载 shader2
        canvas.drawRect(10, 90, 100, 150, mPaint);      //绘制长方形
        RectF rf = new RectF(120, 90, 200, 150);        //实例化 RectF 对象
        mPaint.setShader(shader3);                      //加载 shader3
        canvas.drawRoundRect(rf, 10, 10, mPaint);       //绘制圆角矩形
        rf = new RectF(10, 180, 90, 220);               //实例化 RectF 对象
        canvas.drawOval(rf, mPaint);                    //绘制椭圆
        Path path = new Path();                         //实例化 Path
        path.moveTo(110, 180);                          //定义起始点
        path.lineTo(110, 240);                          //连接点
        path.lineTo(170, 240);                          //连接点
        path.close();                                   //闭合连线
        canvas.drawPath(path, mPaint);                  //绘制多边形
    }

    public int getCount() {
        return count;                                   //返回类别控制变量
    }

    public void setCount(int count) {
        this.count = count;                             //设置类别控制变量
    }
}
```

代码说明：

- 作为绘图画笔的 Paint 类，提供了各种方法来设置绘画的效果。该类的 setStyle() 方法是用来设置描绘内容的风格，参数是由该类的嵌套类 Paint.Style 提供的常量值。Paint.Style.STROKE 表示描绘空心的图形，Paint.Style.FILL 表示描绘实心的图

形，Paint.Style.StrokeAndFill 表示带边框的实心图形。

- 如果设置了风格为 Paint.Style.STROKE 或 Paint.Style.StrokeAndFill，就需要调用 Paint 类的 setStrokeWidth()方法来设置边框的宽度。
- android.graphics.Shader 类是为描绘对象着色的着色器，加载着色器的 Paint 对象将按照着色器的设置绘制图形的颜色。Paint 对象使用 setShader()方法加载着色器。
- android.graphics.LinearGradient 可以实现线性色彩渐变，有两种构造方法。本例中采用的构造方带有 7 个参数。第 1 个参数是 float 类型，表示渐变线的起始 x 坐标；第 2 个参数是 float 类型，表示渐变线的起始 y 坐标；第 3 个参数是 float 类型，表示渐变线的终止 x 坐标；第 4 个参数是 float 类型，表示渐变线的终止 y 坐标；第 5 个参数是 int 类型，表示渐变起始颜色；第 6 个参数是 int 类型，表示渐变终止的颜色；第 7 个参数是 TileMode 类型，表示渐变完成效果。
- LinearGradient 也可以实现多种颜色的渐变，这需要使用它的另外一种构造方法，该构造方法有 7 个参数。第 1 个参数是 float 类型，表示渐变线的起始 x 坐标；第 2 个参数是 float 类型，表示渐变线的起始 y 坐标；第 3 个参数是 float 类型，表示渐变线的终止 x 坐标；第 4 个参数是 float 类型，表示渐变线的终止 y 坐标；第 5 个参数是 int 数组形式，表示参与渐变的各种颜色；第 6 个参数是 float 数组形式，表示每种颜色分布的位置，取值范围 0～1.0，如果参数选择 null，则各种颜色均匀分布；第 7 个参数是 TileMode 类型，表示渐变完成效果。
- 需要注意的是，所谓渐变线的起始坐标和终止坐标不是对于描绘的几何图形，而是相对于容器即本例中的 MySurfaceView 来说的。
- android.graphics.SweepGradient 可以实现 360 度顺时针的颜色渐变，有两种构造方法。本例采用的构造方法带有 4 个参数，第 1 个参数是 float 类型，表示渐变中心的 x 坐标；第 2 个参数是 float 类型，表示渐变中心的 y 坐标；第 3 个参数是 int 类型，表示渐变起始颜色；第 4 个参数是 int 类型，表示渐变终止的颜色。
- SweepGradient 的另一种构造方法可以实现多种颜色的渐变。该构造方法带有 4 个参数，第 1 个参数是 float 类型，表示渐变中心的 x 坐标；第 2 个参数是 float 类型，表示渐变中心的 y 坐标；第 3 个参数是 int 数组形式，表示参与渐变的各种颜色；第 4 个参数是 float 数组形式，表示每种颜色分布的位置，取值范围 0～1.0，如果参数选择 null，则各种颜色均匀分布。
- android.graphics.RadialGradient 可以实现圆弧形的颜色渐变，有两种构造方法。本例采用的构造方法有 6 个参数。第 1 个参数是 float 类型，表示渐变中心的 x 坐标；第 2 个参数是 float 类型，表示渐变中心的 y 坐标；第 3 个参数为 float 类型，为圆的半径；第 4 个参数是 int 类型，表示渐变起始颜色；第 5 个参数是 int 类型，表示渐变终止的颜色；第 6 个参数是 TileMode 类型，表示渐变完成效果。
- RadialGradient 的另一种构造方法可以实现多种颜色的渐变。该构造方法带有 4 个参数，第 1 个参数是 float 类型，表示渐变中心的 x 坐标；第 2 个参数是 float 类型，表示渐变中心的 y 坐标；第 3 个参数为 float 类型，为圆的半径；第 4 个参数是 int 数组形式，表示参与渐变的各种颜色；第 5 个参数是 float 数组形式，表示每种颜色分布的位置，取值范围 0～1.0，如果参数选择 null，则各种颜色均匀分布；第 6 个参数是 TileMode 类型，表示渐变完成效果。

- Canvas 类的 drawCircle()方法用来描绘圆形，该方法带有 4 个参数。第 1 个参数是 float 类型，表示圆心的 x 坐标；第 2 个参数是 float 类型，表示圆心的 y 坐标；第 3 个参数为 float 类型，为圆的半径；第 4 个参数为 Paint 类型，即定义的 Paint 对象。
- Canvas 类的 drawRect()方法用来绘制矩形，该方法有多种重载方式，本例中采用的方法带有 5 个参数。第 1 个参数是 float 类型，表示矩形左上角的 x 坐标；第 2 个参数是 float 类型，表示矩形左上角的 y 坐标；第 3 个参数是 float 类型，表示矩形右下角的 x 坐标；第 4 个参数是 float 类型，表示矩形右下角的 y 坐标；第 5 个参数为 Paint 类型，即定义的 Paint 对象。
- Canvas 类的 drawRoundRect()方法用来绘制圆角矩形，该方法带有 4 个参数。第 1 个参数是 android.graphics.RectF 类型，即定义的 RectF 对象；第 2 个参数是 float 对象，表示圆角的 x 半径；第 3 个参数是 float 对象，表示圆角的 y 半径；第 4 个参数为 Paint 类型，即定义的 Paint 对象。
- android.graphics.RectF 类用来定义矩形坐标，有多种构造方法。本例采用的构造方法有 4 个参数，第 1 个参数是 float 类型，表示矩形左上角的 x 坐标；第 2 个参数是 float 类型，表示矩形左上角的 y 坐标；第 3 个参数是 float 类型，表示矩形右下角的 x 坐标；第 4 个参数是 float 类型，表示矩形右下角的 y 坐标。
- Canvas 类的 drawOval()方法用来描绘椭圆，该方法带两个参数。第 1 个参数是 RectF 类型，即定义的 RectF 对象；第 2 个参数为 Paint 类型，即定义的 Paint 对象。
- Canvas 类的 drawPath()方法用来绘制多边形，该方法有两个参数。第 1 个参数是 android.graphics.Path 类型，即定义的 Path 对象；第 2 个参数为 Paint 类型，即定义的 Paint 对象。
- Path 类用来封装几何图形的线段路径。
- Path 类的 moveTo()方法用来标示一个几何图形的起点，该方法带两个参数，第 1 个参数是 float 类型，表示点的 x 坐标；第 2 个参数是 float 类型，表示点的 y 坐标。
- Path 类的 lineTo()方法用来表示一条线段的终止位置，该方法带两个参数，第 1 个参数是 float 类型，表示终止点的 x 坐标；第 2 个参数是 float 类型，表示终止点的 y 坐标。
- 如果调用 lineTo()方法之前没有调用过 moveTo()方法，则从（0，0）点开始描绘线段。
- Path 类的 close()方法用来闭合当前多边形。如果当前多边形的最后一个点不是描绘的第一个点，则会自动地在第一个点和最后一个点之间描绘一条线段。

接下来是调用 MySurfaceView 的 Activity 类，代码如下：

```
public class MyActivity extends Activity{
    private Button button1;                 //定义 Button 对象
    private Button button2;                 //定义 Button 对象
    private Button button3;                 //定义 Button 对象
    private MySurfaceView mySurfaceView;    //定义 MySurfaceView 对象
   public void onCreate(Bundle savedInstanceState){
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);       //加载布局 xml 文件
       mySurfaceView=(MySurfaceView)this.findViewById(R.id.sview);
```

```
                                    //获取 MySurfaceView 对象实例
    button1=(Button)this.findViewById(R.id.button1);
                                    //获取 Button 对象实例
    button1.setOnClickListener(new OnClickListener(){
                                    //为 Button 对象添加单击事件监听
        public void onClick(View v) {
            mySurfaceView.setCount(0);  //设置 MySurfaceView 对象控制变量
        }
     });

    button2=(Button)this.findViewById(R.id.button2);
                                    //获取 Button 对象实例
    button2.setOnClickListener(new OnClickListener(){
                                    //为 Button 对象添加单击事件监听
        public void onClick(View v) {
            mySurfaceView.setCount(1);//设置 MySurfaceView 对象控制变量
        }
     });

    button3=(Button)this.findViewById(R.id.button3);
                                    //获取 Button 对象实例
    button3.setOnClickListener(new OnClickListener(){
                                    //为 Button 对象添加单击事件监听
        public void onClick(View v) {
            mySurfaceView.setCount(2);  //设置 MySurfaceView 对象控制变量
        }
     });
  }
}
```

程序运行后，单击“图形”按钮，绘制各种空心几何图形，效果如图 10.4 所示。单击“填充”按钮，绘制各种填充颜色的几何图形，如图 10.5 所示。

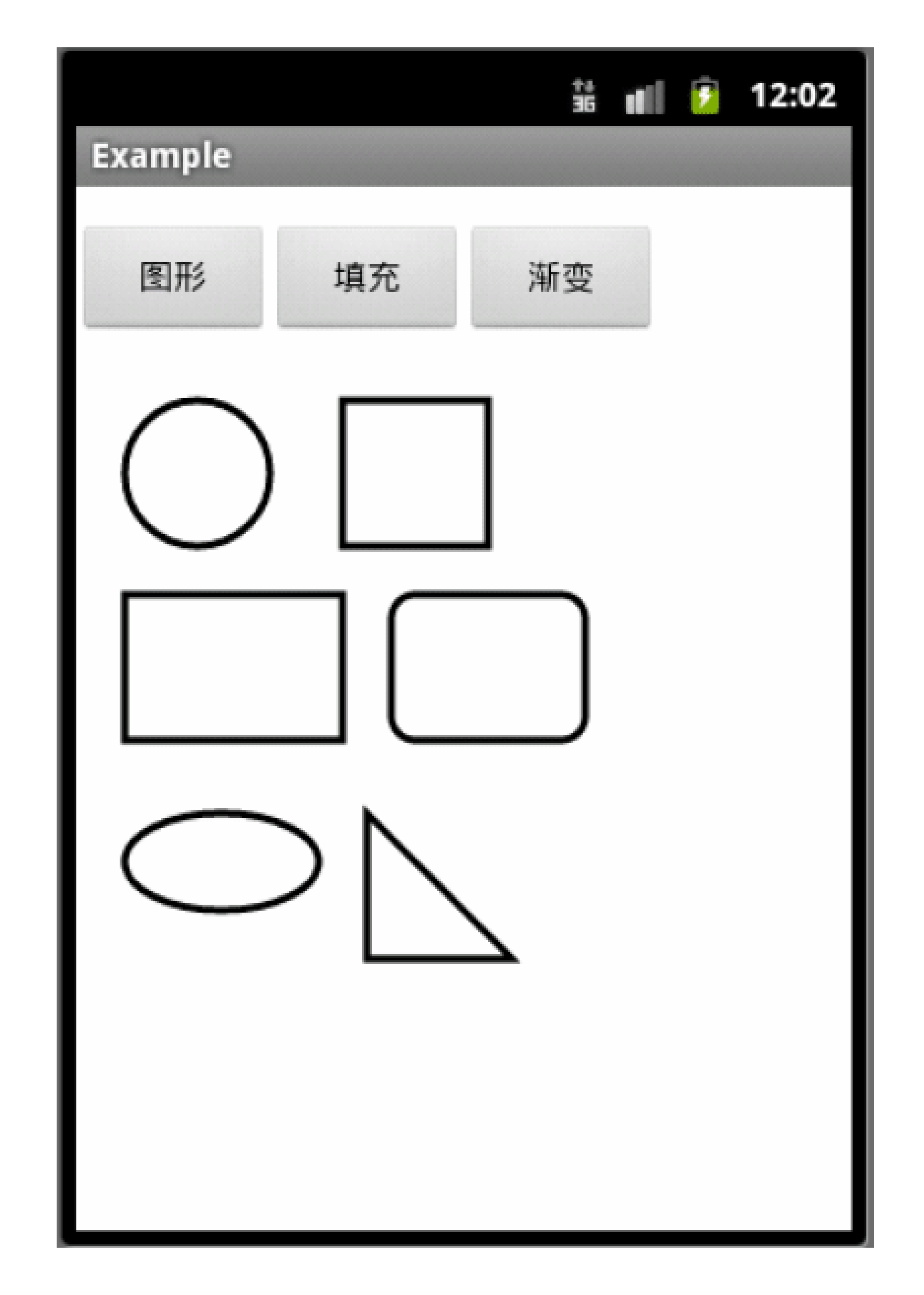


图 10.4　空心几何图形

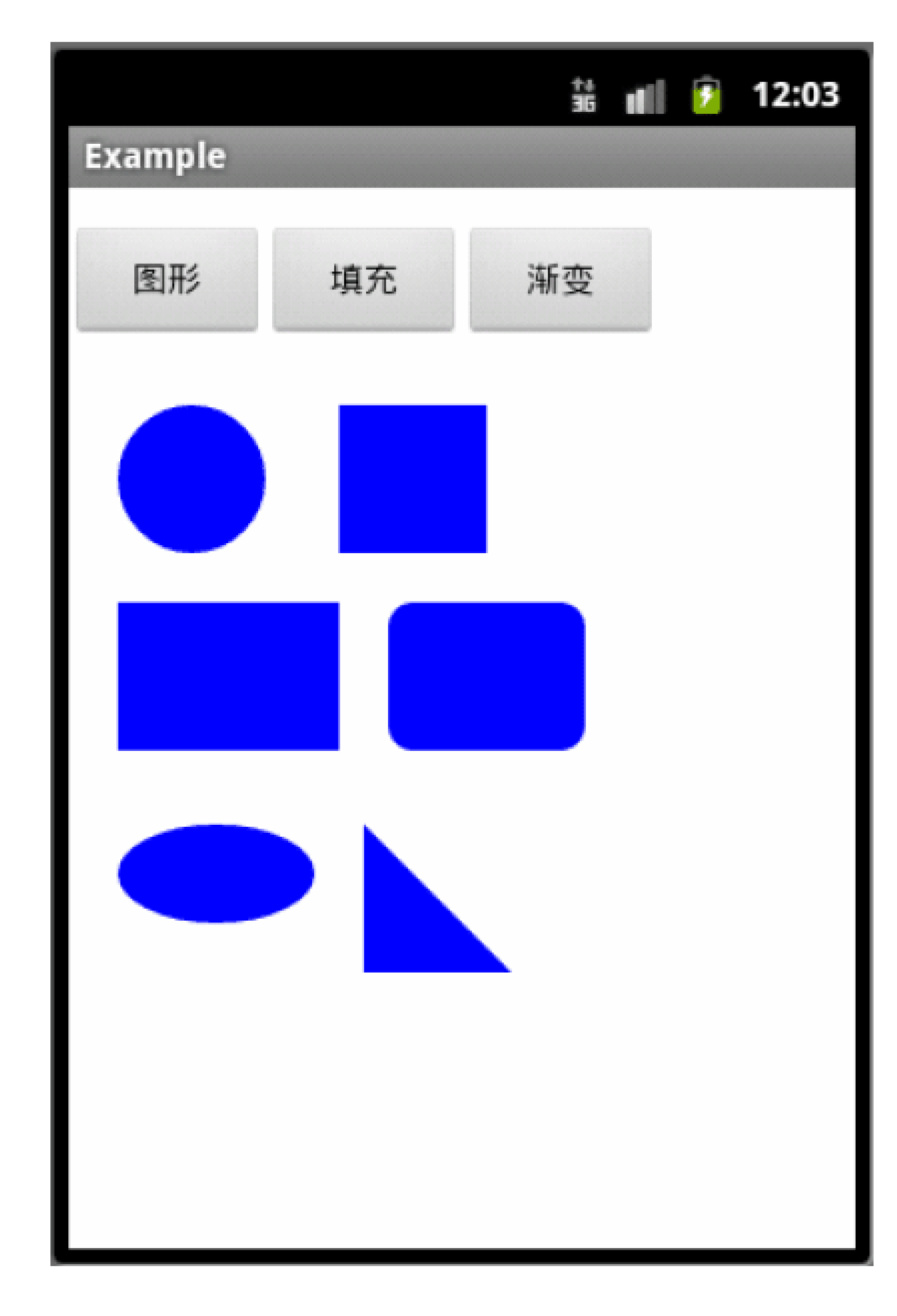


图 10.5　填充色几何图形

单击“渐变”按钮，描绘各种渐变色几何图形。其中，圆形和正方形使用 LinearGradient 渐变，长方形使用 SweepGradient 渐变，圆角矩形、椭圆形和三角形使用 RadialGradient 渐变，效果如图 10.6 所示。

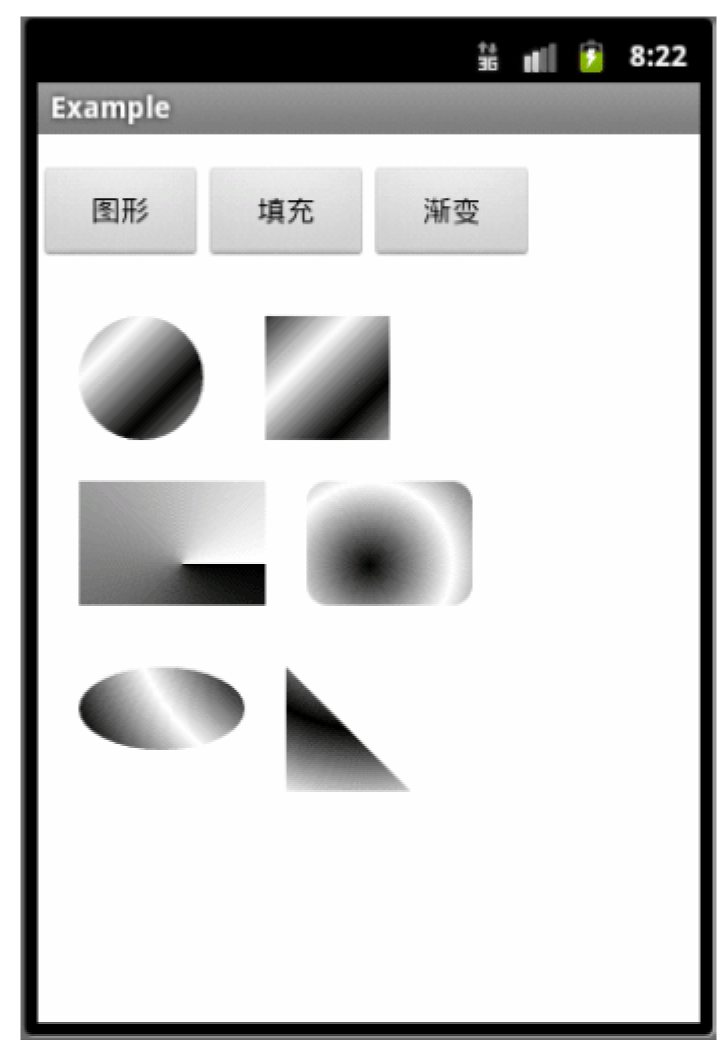


图 10.6　渐变色几何图形

10.4　Matrix 对象处理图像

除了几何图形，Canvas 还可以描绘图像。在实际应用中，经常会有对图像进行特殊处理的需求。在 Android SDK 中，可以使用 Matrix 类来处理这些需求。本节中就来研究如何使用 Matrix 类缩放、旋转和倾斜图像。首先是 xml 布局文件，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
                                            <!-- 线形布局 -->
    android:orientation="vertical" android:layout width="fill parent"
                                            <!--垂直-->
    android:layout height="fill parent" android:background="#ffffffff">
    <LinearLayout android:orientation="horizontal"     <!--线形布局-->
        android:layout width="wrap content" android:layout height=
        "wrap_content"      <!--水平-->
        android:layout_marginBottom="15px">
        <Button android:id="@+id/button1" android:layout_width="80px"
                                            <!--定义按钮-->
            android:layout_height="wrap_content" android:text="缩放"
            android:layout_marginTop="15px" />
        <Button android:id="@+id/button2" android:layout_width="80px"
                                            <!--定义按钮-->
            android:layout height="wrap content" android:text="旋转"
            android:layout marginTop="15px" />

        <Button android:id="@+id/button3" android:layout_width="80px"
                                            <!--定义按钮-->
            android:layout_height="wrap_content" android:text="倾斜"
```

```
            android:layout marginTop="15px" />

        <Button android:id="@+id/button4" android:layout width="80px"
                                    <!--定义按钮-->
            android:layout_height="wrap_content" android:text="还原"
            android:layout_marginTop="15px" />

    </LinearLayout>
    <com.example.MySurfaceView android:id="@+id/sview" <!--自定义控件-->
        android:layout gravity="center" android:layout width="300px"
        android:layout height="300px" />
</LinearLayout>
```

代码说明：

界面布局中总体是一个垂直线形布局，布局中上半部分是一个水平线形布局，包含 4 个按钮；下半部分是一个我们自定义继承 SurfaceView 的类。

接下来是继承了 SurfaceView 类的 MySurfaceView 类，实现显示图像、缩放图像和旋转图像的功能，代码如下：

```
public class MySurfaceView extends SurfaceView implementsSurfaceHolder.
Callback, Runnable {
    private SurfaceHolder mSurfaceHolder = null;  //定义 SurfaceHolder 对象
    private int count = -1;                      //定义类别控制变量
    private boolean pan = true;                  //定义循环控制变量
    private Context cot;                         //定义 Context 变量
    private Bitmap bitmap = null;                //定义 Bitmap 变量
    private float zoom=1.0f;                     //定义缩放率变量
    private boolean zoompan=true;                //定义缩放判断变量
    private float rotate=0;                      //定义旋转角度变量

    public MySurfaceView(Context context, AttributeSet attrs) {
        super(context, attrs);                   //调用父类构造方法
        cot = context;                           //为 Context 对象赋值
        mSurfaceHolder = this.getHolder();       //获取 SurfaceHolder 对象实例
        mSurfaceHolder.addCallback(this); //为 SurfaceHolder 对象实例添加回调
    }

    public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {
    }                              // SurfaceView 对象实例尺寸改变时触发

    public void surfaceCreated(SurfaceHolder holder) {
        count=0;                   //控制变量赋值
        new Thread(this).start();  //开始新的线程
    }

    public void surfaceDestroyed(SurfaceHolder holder) {
                                   //SurfaceView 对象实例销毁时触发
        count = -1;                //初始化类别控制变量
        pan = false;               //初始化循环控制变量
    }

    public void run() {            //线程工作方法
        while (pan) {              //循环条件
            synchronized (mSurfaceHolder) {     //同步 mSurfaceHolder 对象
                Canvas canvas = mSurfaceHolder.lockCanvas();
```

```
                                    //锁定并获得 Canvas 类对象
                canvas.drawColor(Color.WHITE); //绘制背景色
                Matrix matrix = new Matrix();  //定义 Matrix 对象
                switch (this.getCount()) {     //判断绘制条件
                case 0:
                    drawPic(matrix);           //绘制原始图片
                    break;
                case 1:
                    zoomPic(matrix);           //缩放图片
                    break;
                case 2:
                    rotatePic(matrix);         //旋转图片
                    break;
                case 3:
                    skewPic(matrix);           //倾斜图片
                    break;
                }
                canvas.setDrawFilter(new PaintFlagsDrawFilter(0,
                                           //Canvas 设置抗锯齿
                      Paint.ANTI_ALIAS_FLAG|Paint.FILTER_BITMAP_FLAG));
                canvas.drawBitmap(bitmap, matrix, null);   //描绘图片
                mSurfaceHolder.unlockCanvasAndPost(canvas);
                                           //解锁并显示描绘内容
            }
            try {
                Thread.sleep(1000);        //线程休眠 1 000 毫秒
            } catch (Exception e) {
            }
        }
    }

    public void drawPic(Matrix matrix){
        try {
            InputStream iso = cot.getAssets().open("pic/p1.jpg");
                                           //读取 assets 文件夹下图片
            bitmap = BitmapFactory.decodeStream(iso);
                                           //生成 Bitmap 对象
        } catch (Exception e) {
            e.printStackTrace();
        }
        zoom=0.8f;                         //初始化缩放变量
        rotate=0;                          //初始化旋转变量
    }

    public void zoomPic(Matrix matrix){
        if(zoom<0.3 && zoompan){           //判断缩小下限
            zoompan=false;                 //更改缩放判断变量
        }
        if(zoom>0.9 && !zoompan){          //判断放大上限
            zoompan=true;                  //更改缩放判断变量
        }
        if(zoompan){
            zoom=zoom-0.1f;                //增大缩放比率
        }
        else{
            zoom=zoom+0.1f;                //减小缩放比率
        }
```

```
        matrix.setScale(zoom, zoom);        //设置缩放比率
        matrix.postTranslate(300 / 2 - bitmap.getWidth()*zoom / 2,
                300 / 2 - bitmap.getHeight()*zoom / 2 );    //移动图像位置
    }

    public void rotatePic(Matrix matrix){
        rotate=rotate+60;                   //累加旋转角度
        matrix.setScale(zoom, zoom);        //设置缩放比率
        matrix.postTranslate(300 / 2 - bitmap.getWidth()*zoom / 2,
                300 / 2 - bitmap.getHeight()*zoom / 2 ); //移动图像位置
        matrix.postRotate(rotate,300 / 2, 300 / 2);     //旋转图像
    }

    public void skewPic(Matrix matrix){
        matrix.setScale(zoom, zoom);        //设置缩放比率
        matrix.postTranslate(300 / 2 - bitmap.getWidth()*zoom / 2,
                300 / 2 - bitmap.getHeight()*zoom / 2 ); //移动图像位置
        matrix.postSkew(0.1f,0.1f,300 / 2, 300 / 2);
                                            //设置倾斜角度及中心坐标
    }

    public int getCount() {
        return count;                       //返回类别控制变量
    }

    public void setCount(int count) {
        this.count = count;                 //设置类别控制变量
    }
}
```

代码说明：

- android.graphics. Matrix 类为每一种图像操作提供了 3 种不同前缀的方法，分别是 set、post 和 pre。当需要初始化一个 Matrix 对象的时候，需要使用前缀为 set 的方法；如果要对一个图像进行多种操作，则后续的操作可以使用前缀为 post 或 pre 的方法。
- Matrix 类的 scale 操作可以缩放图像，该方法有两种重载方式，本例中使用的方法带两个参数。第 1 个参数为 float 类型，表示 x 轴缩放的比率；第 2 个参数为 float 类型，表示 y 轴的缩放比率。
- Matrix 类的 translate 操作可以平移图像，该方法带两个参数。第 1 个参数为 float 类型，表示平移后图像中心的 x 坐标；第 2 个参数为 float 类型，表示平移后图像中心的 y 坐标。
- Matrix 类的 rotate()方法可以旋转图像，该方法有两种重载方式，本例中使用的方法带有 3 个参数。第 1 个参数为 float 类型，表示旋转角度，顺时针方向增大；第 2 个参数为 float 类型，表示旋转中心点的 x 轴坐标；第 3 个参数为 float 类型，表示旋转中心点的 y 轴坐标。
- Matrix 类的 skew()方法可以倾斜图像，该方法有两种重载方式，本例中使用的方法带有 4 个参数。第 1 个参数为 float 类型，表示 x 轴倾斜角度；第 2 个参数为 float 类型，表示 y 轴倾斜角度；第 3 个参数为 float 类型，表示倾斜中心点的 x 轴坐标；第 4 个参数为 float 类型，表示倾斜中心点的 y 轴坐标。

接下来是调用 MySurfaceView 的 Activity 类，代码如下：

```
public class MyActivity extends Activity{
    private Button button1;                  //定义 Button 对象
    private Button button2;                  //定义 Button 对象
    private Button button3;                  //定义 Button 对象
    private Button button4;                  //定义 Button 对象
    private MySurfaceView mySurfaceView;     //定义 MySurfaceView 对象
   public void onCreate(Bundle savedInstanceState){
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);         //加载布局 xml 文件
      mySurfaceView=(MySurfaceView)this.findViewById(R.id.sview);
                                             //获取 MySurfaceView 对象实例
      button1=(Button)this.findViewById(R.id.button1);
                                             //获取 Button 对象实例
      button1.setOnClickListener(new OnClickListener(){
                                             //为 Button 对象添加单击事件监听
          public void onClick(View v) {
              mySurfaceView.setCount(1);  //设置 MySurfaceView 对象控制变量
          }
       });

      button2=(Button)this.findViewById(R.id.button2);
                                             //获取 Button 对象实例
      button2.setOnClickListener(new OnClickListener(){
                                             //为 Button 对象添加单击事件监听
          public void onClick(View v) {
              mySurfaceView.setCount(2);//设置 MySurfaceView 对象控制变量
          }
       });

      button3=(Button)this.findViewById(R.id.button3);
                                             //获取 Button 对象实例
      button3.setOnClickListener(new OnClickListener(){
                                             //为 Button 对象添加单击事件监听
          public void onClick(View v) {
              mySurfaceView.setCount(3);  //设置 MySurfaceView 对象控制变量
          }
       });

      button4=(Button)this.findViewById(R.id.button4);
                                             //获取 Button 对象实例
      button4.setOnClickListener(new OnClickListener(){
                                             //为 Button 对象添加单击事件监听
          public void onClick(View v) {
              mySurfaceView.setCount(0);  //设置 MySurfaceView 对象控制变量
          }
       });
   }
}
```

程序运行后，单击“缩放”按钮后，图像会在一定范围内交替实现缩小、放大的效果，如图 10.7 所示。单击“旋转”按钮后，图像会以当前缩放级别进行顺时针旋转，如图 10.8 所示。

单击“倾斜”按钮后，图像会以当前缩放级别，以图像中心为中心倾斜，如图 10.9 所示。




图 10.7　图像缩放

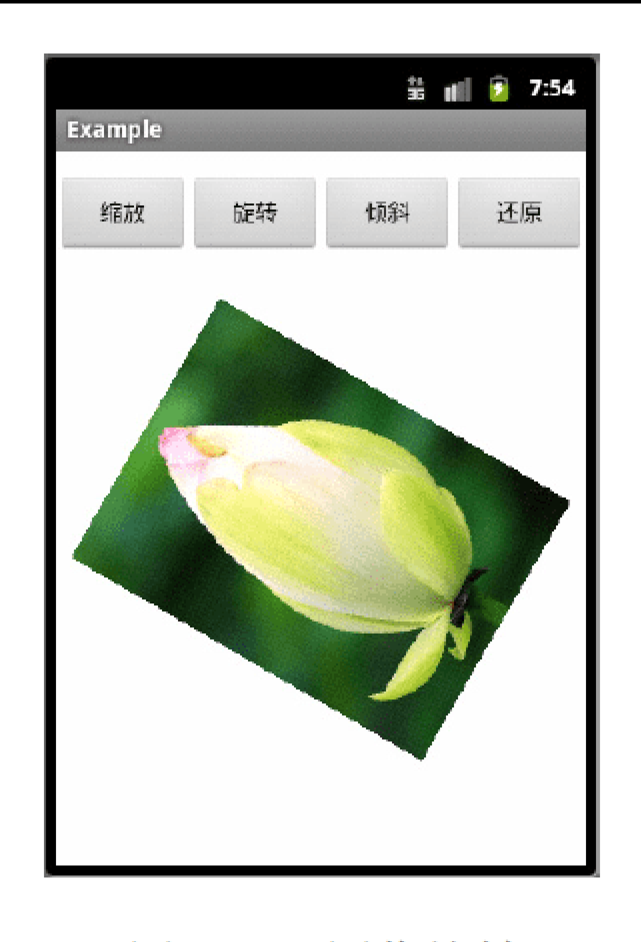


图 10.8　图像旋转



图 10.9　图像倾斜

10.5　动 画 处 理

在上一节的示例中，虽然有一些动画的“效果”，但其实并不是 Android SDK 中真正意义上的动画。Android SDK 中有两种处理动画的方式，即 Frame 方式和 Tween 方式，本节将分别讲解这两种方式。

10.5.1　Frame 动画

前面的例子中，曾使用不断加载顺序图片的形式来形成动画效果。Android SDK 中的 Frame 动画也是按照这种机制完成的，这种机制的全称就是 frame-by-frame animations。

要实现 Frame 动画首先要准备一个 xml，可以在 res 文件夹下创建一个文件夹，命名为 anim，然后添加一个名为 animation 的 xml 文件，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
                                        <!--定义 animation-->
    android:oneshot="false">
    <item android:drawable="@drawable/p0" android:duration="200" />
                                        <!--添加渐变图片-->
    <item android:drawable="@drawable/p1" android:duration="200" />
                                        <!--添加渐变图片-->
    <item android:drawable="@drawable/p2" android:duration="200" />
                                        <!--添加渐变图片-->
    <item android:drawable="@drawable/p3" android:duration="200" />
                                        <!--添加渐变图片-->
    <item android:drawable="@drawable/p4" android:duration="200" />
                                        <!--添加渐变图片-->
    <item android:drawable="@drawable/p5" android:duration="200" />
                                        <!--添加渐变图片-->
    <item android:drawable="@drawable/p6" android:duration="200" />
                                        <!--添加渐变图片-->
    <item android:drawable="@drawable/p7" android:duration="200" />
                                        <!--添加渐变图片-->
```

```
    <item android:drawable="@drawable/p8" android:duration="200" />
                                   <!--添加渐变图片-->
    <item android:drawable="@drawable/p9" android:duration="200" />
                                   <!--添加渐变图片-->
    <item android:drawable="@drawable/p10" android:duration="200" />
                                   <!--添加渐变图片-->
    <item android:drawable="@drawable/p11" android:duration="200" />
                                   <!--添加渐变图片-->
</animation-list>
```

代码说明：

- <animation-list>标签的 oneshot 属性表示动画是否重复播放，false 表示重复播放，true 表示不重复播放。
- <item>标签的 drawable 属性设置资源位置，duration 属性设置这一帧动画播放时间。

然后准备 xml 布局文件，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:background="#ffffffff">
    <Button android:id="@+id/button" android:layout_width="80px"
        android:layout_height="wrap_content" android:text="播放"
        android:layout_marginTop="15px" />
    <ImageView android:id="@+id/view" android:layout_marginTop="15px"
        android:layout_gravity="center" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:src="@anim/animation" />
</LinearLayout>
```

代码说明：

< ImageView >标签的 src 属性加载准备好的 animation.xml。

最后，只需要在 Activity 中调用就可以了，MyActivity.java 代码如下：

```
public class MyActivity extends Activity{
    private Button button;                  //定义 Button 对象
    private ImageView view;                 //定义 ImageView 对象
    private AnimationDrawable draw;         //定义 AnimationDrawable 对象
   public void onCreate(Bundle savedInstanceState){
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);        //加载 xml 布局文件
      view=(ImageView)this.findViewById(R.id.view);//实例化 ImageView 对象
      draw=(AnimationDrawable)view.getDrawable();
                                            //实例化 AnimationDrawable 对象
      button=(Button)this.findViewById(R.id.button);  //实例化 Button 对象
      button.setOnClickListener(new OnClickListener(){
                                            //Button 对象加载监听事件
           public void onClick(View v) {
               if(draw.isRunning()){
                                //判断动画是否播放
                   draw.stop();//停止动画
               }
               draw.start();    //播放动画
           }
       });
    }
}
```

程序运行效果如图 10.10 所示。

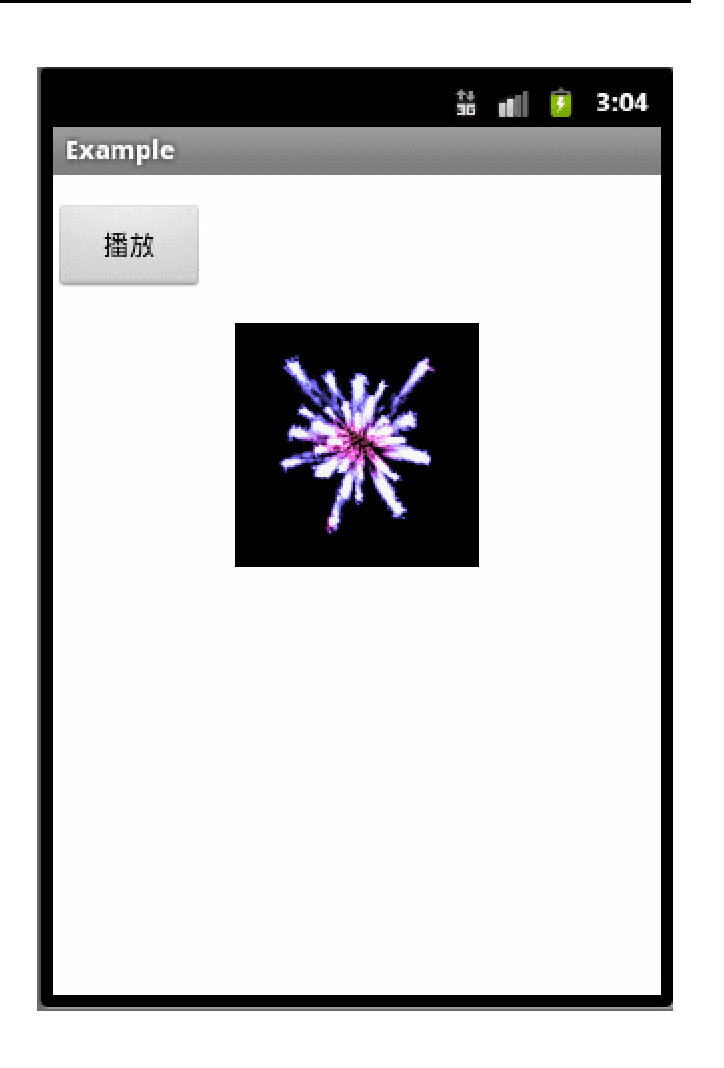


图 10.10　Frame 动画效果

10.5.2　Tween 动画

Tween 动画类型又称为“补间动画”，可以实现 4 种动画效果，分别是 Alpha 渐变透明度动画效果、Scale 渐变尺寸伸缩动画效果、Translate 画面转换位置移动动画效果和 Rotate 画面旋转动画效果。Tween 动画可以通过 xml 配置文件或者 Java 代码两种方式来实现。

1．Java方式

首先来看一下由纯 Java 代码来完成的效果，下面是一个简单的 Tween 动画实例，代码如下：

```
public class MyView extends View {
    private Bitmap bitmap = null;          //定义 Bitmap 对象
    private Animation animation = null;    //定义 Animation 对象
    public MyView(Context context, AttributeSet attrs) {
        super(context, attrs);
        bitmap = ((BitmapDrawable) getResources().getDrawable(R.drawable.p1))
                .getBitmap();              //加载资源实例化 Bitmap 对象
    }

    protected void onDraw(Canvas canvas) { //重写 onDraw()方法
        super.onDraw(canvas);
        canvas.drawBitmap(bitmap, 0, 0, null);     //绘制图像
    }

    public void type(int num) {
        switch (num) {                     //判断类型
        case 0:
            animation = new AlphaAnimation(1.0f, 0.1f);
                                           //实例化 AlphaAnimation 对象
            animation.setDuration(3000);   //设置动画持续时间
            animation.setRepeatMode(2);    //设置动画重复模式
            animation.setRepeatCount(3);   //设置动画重复次数
            this.startAnimation(animation);//开始动画
            break;
        case 1:
            animation = new ScaleAnimation(1.0f, 0.0f, 1.0f, 0.0f,
                    Animation.RELATIVE TO SELF, 0.5f,
                    Animation.RELATIVE TO SELF, 0.5f);
                                           //实例化 ScaleAnimation 对象
            animation.setDuration(1000);   //设置动画持续时间
            animation.setRepeatMode(2);    //设置动画重复模式
            animation.setRepeatCount(3);   //设置动画重复次数
            this.startAnimation(animation);//开始动画
            break;
        case 2:
            animation = new TranslateAnimation(10, 100, 10, 100);
                                       //实例化 TranslateAnimation 对象
            animation.setDuration(1000);   //设置动画持续时间
```

```
            animation.setRepeatMode(2);         //设置动画重复模式
            animation.setRepeatCount(3);        //设置动画重复次数
            this.startAnimation(animation);     //开始动画
            break;
        case 3:
            animation = new RotateAnimation(0.0f, +360,
                    Animation.RELATIVE TO SELF, 0.5f,
                    Animation.RELATIVE TO SELF, 0.5f);
                                                //实例化RotateAnimation对象
            animation.setDuration(1000);        //设置动画持续时间
            animation.setStartOffset(1000);     //设置动画间隔时间
            animation.setRepeatMode(1);         //设置动画重复模式
            animation.setRepeatCount(4);        //设置动画重复次数
            this.startAnimation(animation);     //开始动画
            break;
        }
    }
}
```

代码说明：

- android.view.animation.Animation 类为动画的抽象基类，定义了各种动画效果的通用方法。
- android.view.animation.AlphaAnimation 是实现透明渐变动画的类，有两种重载的构造方法，本例中使用的构造方法带有两个参数。第 1 个参数是 float 类型，表示渐变开始时的 alpha 值；第 2 个参数是 float 类型，表示渐变结束时的 alpha 值。
- alpha 值的取值范围是从 1.0～0.0 的小数。1.0 表示完全不透明，0.0 表示完全透明。
- Animation 类的 setDuration()方法设置动画持续的时间，参数是 long 类型，表示动画持续的毫秒数。
- Animation 类的 setRepeatMode()方法设置动画重复的方式，该方法的参数是一个整数类型，取值范围是 Animation 类的两个常量值，即 Animation.RESTART 和 Animation. REVERSE。
- Animation.RESTART 表示重新播放动画效果，常量值为 1；Animation. REVERSE 表示反向播放动画效果，常量值为 2。
- Animation 类的 setRepeatCount()方法设置动画重复播放的次数，参数为整数类型。
- android.view.animation.ScaleAnimation 是实现尺寸伸缩动画的类，有多种重载的构造方法，本例中采用的方法带有 8 个参数。第 1 个参数为 float 类型，表示动画开始时水平方向缩放系数；第 2 个参数是 float 类型，表示动画结束时水平方向缩放系数；第 3 个参数是 float 类型，表示动画开始时垂直方向缩放系数；第 4 个参数是 float 类型，表示动画结束时垂直方向缩放系数；第 5 个参数是 int 类型，表示水平方向处理缩放数值的方式，可以选择的值有 Animation.ABSOLUTE（按一个绝对像素值缩放）、Animation.RELATIVE_TO_SELF（以自身的高宽缩放）、Animation.RELATIVE_TO_PARENT（以容器的高宽缩放）；第 6 个参数为 float 类型，表示开始缩放时水平方向的中心点坐标，如果水平方向处理缩放数值的方式选择了 Animation.ABSOLUTE，则需要一个正数，如果水平方向处理缩放数值的方式选择了 Animation.RELATIVE_TO_SELF 或 Animation.RELATIVE_TO_PARENT，则需要填写一个 0.0～1.0 之间的小数；第 7 个参数是 int 类型，表示垂直方向处理缩放数值的方式，可以选择的值有 Animation.ABSOLUTE（按一个绝

对像素值缩放）、Animation.RELATIVE_TO_SELF（以自身的高宽缩放）、Animation.RELATIVE_TO_PARENT（以容器的高宽缩放）；第 8 个参数为 float 类型，表示开始缩放时垂直方向的中心点坐标，如果垂直方向处理缩放数值的方式选择了 Animation.ABSOLUTE，则需要一个正数，如果垂直方向处理缩放数值的方式选择了 Animation.RELATIVE_TO_SELF 或 Animation.RELATIVE_TO_PARENT，则需要填写一个 0.0～1.0 之间的小数。

- android.view.animation.TranslateAnimation 是实现位置移动动画效果的类，有多种重载的构造方法，本例中采用的方法带有 4 个参数。第 1 个参数是 float 类型，表示动画开始是水平坐标的位置；第 2 个参数是 float 类型，表示动画结束时，水平坐标的位置；第 3 个参数是 float 类型，表示动画开始是垂直坐标的位置；第 4 个参数是 float 类型，表示动画结束时垂直坐标的位置。
- android.view.animation.RotateAnimation 是实现画面旋转动画效果的类，有多种重载的构造方法，本例中采用的方法带有 6 个参数。第 1 个参数是 float 类型，表示动画开始时的角度；第 2 个参数是 float 类型，表示动画结束时的角度；第 3 个参数是 int 类型，表示水平方向处理缩放数值的方式，可以选择的值有 Animation.ABSOLUTE（按一个绝对像素值缩放）、Animation.RELATIVE_TO_SELF（以自身的高宽缩放）、Animation.RELATIVE_TO_PARENT（以容器的高宽缩放）；第 4 个参数为 float 类型，表示开始缩放时水平方向的中心点坐标，如果水平方向处理缩放数值的方式选择了 Animation.ABSOLUTE，则需要一个正数，如果水平方向处理缩放数值的方式选择了 Animation.RELATIVE_TO_SELF 或 Animation.RELATIVE_TO_PARENT，则需要填写一个 0.0～1.0 之间的小数；第 5 个参数是 int 类型，表示垂直方向处理缩放数值的方式，可以选择的值有 Animation.ABSOLUTE（按一个绝对像素值缩放）、Animation.RELATIVE_TO_SELF（以自身的高宽缩放）、Animation.RELATIVE_TO_PARENT（以容器的高宽缩放）；第 6 个参数为 float 类型，表示开始缩放时垂直方向的中心点坐标，如果垂直方向处理缩放数值的方式选择了 Animation.ABSOLUTE，则需要一个正数，如果垂直方向处理缩放数值的方式选择了 Animation.RELATIVE_TO_SELF 或 Animation.RELATIVE_TO_PARENT，则需要填写一个 0.0～1.0 之间的小数。
- Animation 类的 setStartOffset()方法设置动画播放的间隔时间，参数为 long 类型。
- android.view.View 类的 startAnimation()方法用来启动该对象的动画效果，参数就是定义的 Animation 类对象。

然后就是调用的 Activity 类，代码如下：

```
public class MyActivity extends Activity {
    private Button button1;                 //定义 Button 对象
    private Button button2;                 //定义 Button 对象
    private Button button3;                 //定义 Button 对象
    private Button button4;                 //定义 Button 对象
    private MyView myView;                  //定义 MyView 对象
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);      //加载 xml 布局文件
        myView=(MyView)this.findViewById(R.id.sview);  //实例化 MyView 对象

       button1=(Button)this.findViewById(R.id.button1);//实例化 Button 对象
```

```
        button1.setOnClickListener(new OnClickListener(){ //加载事件监听
             public void onClick(View v) {
                 myView.type(1);                //调用方法
             }
         });

        button2=(Button)this.findViewById(R.id.button2);//实例化 Button 对象
        button2.setOnClickListener(new OnClickListener(){//加载事件监听
             public void onClick(View v) {
                 myView.type(2);                //调用方法
             }
         });

        button3=(Button)this.findViewById(R.id.button3);//实例化 Button 对象
        button3.setOnClickListener(new OnClickListener(){//加载事件监听
             public void onClick(View v) {
                 myView.type(3);                //调用方法
             }
         });

        button4=(Button)this.findViewById(R.id.button4);//实例化 Button 对象
        button4.setOnClickListener(new OnClickListener(){ //加载事件监听
             public void onClick(View v) {
                 myView.type(0);                //调用方法
             }
         });
    }
}
```

运行程序后，单击“缩放”按钮，图像会以自身的中心点为中心缩小，缩小到图像消失后，再进行放大，如此循环 3 次，效果如图 10.11 所示；单击“位移”按钮后，图像会向右下角平移，然后返回，重复 3 次，效果如图 10.12 所示；单击“旋转”按钮后，图像以自身的中心点为中心旋转 360 度，重复 3 次，每次间隔一秒钟，效果如图 10.13 所示；单击“渐变”按钮后，图像会逐渐透明至消失，然后逐渐复原，反复 3 次，效果如图 10.14 所示。



图 10.11　缩放动画效果



图 10.12　位移动画效果



图 10.13　旋转动画效果



图 10.14　渐变动画效果

2．XML配置方式

要使用 xml 文件配置实现上面例子中同样的动画效果，需要创建动画定义文件。alpha 动画配置文件如下。

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <alpha android:fromAlpha="1.0" android:toAlpha="0.1"
        android:duration="3000" android:repeatMode="reverse" android:
        repeatCount="3" />
</set>
```

rotate 动画配置文件如下。

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <rotate android:interpolator="@android:anim/accelerate decelerate
    interpolator"
        android:fromDegrees="0" android:toDegrees="+360" android:pivotX=
        "50%"android:pivotY="50%" android:duration="1000" android:
        startOffset="1000"android:repeatMode="restart" android:
        repeatCount="3" />
</set>
```

scale 动画配置文件如下。

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <scale android:fromXScale="1.0" android:toXScale="0.0"
        android:fromYScale="1.0" android:toYScale="0.0" android:pivotX=
        "50%" android:pivotY="50%" android:duration="1000"android:
        repeatMode="reverse" android:repeatCount="3" />
</set>
```

translate 动画配置文件如下。

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
```

```
    <translate android:fromXDelta="10" android:toXDelta="100"
        android:fromYDelta="10" android:toYDelta="100" android:duration=
        "1000"android:repeatMode="reverse" android:repeatCount="3" />
</set>
```

上述 4 个动画效果配置文件放在 res 文件夹下的 anim 文件夹中，在需要的时候调用即可，代码如下：

```
public class MyView extends View {
    private Bitmap bitmap = null;           //定义 Bitmap 对象
    private Animation animation = null;     //定义 Animation 对象
    private Context cont=null;              //定义 Context 对象
    public MyView(Context context, AttributeSet attrs) {
        super(context, attrs);
        cont=context;                       //实例化 Context 对象
        bitmap = ((BitmapDrawable) getResources().getDrawable
        (R.drawable.p1))
                .getBitmap();               //实例化 Bitmap 对象
    }

    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        canvas.drawBitmap(bitmap, 0, 0, null); //绘制 Bitmap 图像
    }

    public void type(int num) {
        switch (num) {
        case 0:
            animation = AnimationUtils.loadAnimation(cont,R.anim.alpha);
                                                //加载 alpha 动画配置文件
            this.startAnimation(animation);     //启动动画
            break;
        case 1:
            animation =AnimationUtils.loadAnimation(cont,R.anim.scale);
                                                //加载 scale 动画配置文件
            this.startAnimation(animation);     //启动动画
            break;
        case 2:
           animation =AnimationUtils.loadAnimation(cont,R.anim. translate);
                                                //加载 translate 动画配置文件
            this.startAnimation(animation);     //启动动画
            break;
        case 3:
            animation =AnimationUtils.loadAnimation(cont,R.anim.rotate);
                                                //加载 rotate 动画配置文件
            this.startAnimation(animation);     //启动动画
            break;
        }
    }
}
```

程序运行效果和纯 Java 代码的示例是一致的。

第 11 章　Android 与 Internet

随着手机 3G 时代的到来，手机商务、视频通话、手机音乐、手机游戏等高速数据业务应用越来越多地进入到人们的日常生活当中。Android 当然不会落后于时代潮流，它提供了多种方式可以使编程人员轻松地编写出满足各种网络需求应用程序。在本章中，就来一一了解这些功能。

11.1　程序内置浏览器 WebView

WebView 是 Android 内置的浏览器组件，它将一个 WebKit 内核的浏览器嵌入到应用程序当中，可以使应用程序快速、便捷地访问互联网上的页面。

11.1.1　准备工作

要让应用程序能够访问互联网，就必需要添加相应的权限。在程序的 AndroidManifest.xml 文件中添加如下内容：

```
<uses-permission android:name="android.permission.INTERNET"/>
```

11.1.2　修改布局文件

在 res/layout/main.xml 文件中，添加如下代码，将 WebView 组件元素添加到布局文件的 LinearLayout 布局中。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <WebView android:id="@+id/web1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" />
</LinearLayout>
```

代码说明如下。

- android:id：设置组件的编号。
- android:layout_width：设置组件的宽度。
- android:layout_height：设置组件的高度。

11.1.3　访问互联网页面

同使用其他组件一样，只需要在 Java 文件中声明 WebView 组件的对象并实例化，就可以使用它，代码如下：

```
public class MyWebView extends Activity{
    private WebView mweb;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.main);
        mweb = (WebView) findViewById(R.id.web1);//实例化 WebView 组件对象 mweb
        mweb.loadUrl("http://www.baidu.com");   //加载所需页面的 URL
    }
}
```

效果如图 11.1 所示。



图 11.1　WebView 加载网络页面

11.1.4　访问应用程序内置页面

WebView 组件不光能够加载和互联网上的页面，也可以根据需要加载应用程序中内置的页面。Android 内置了一个前缀为"file:///android_asset/"的结构，WebView 会根据这个结构到应用程序的 assets 文件夹下去寻找加载的页面。先在 assets 文件夹下添加一个测试的页面 index.html 和一张图片，内容如下：

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
    这是程序内置测试页面
    <br/>
    <img src="flower.jpg" />
</body>
```

然后修改 Java 文件中的代码，MyWebView.java 代码如下：

```
public class MyWebView extends Activity{
    private WebView mweb;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.main);

        mweb = (WebView) findViewById(R.id.web1);
        mweb.loadUrl("file:///android_asset/index.html");
                                              //加载 assets 文件夹下页面
    }
}
```

效果如图 11.2 所示。



图 11.2　WebView 加载应用程序内置页面

11.1.5　WebView 页面事件处理

如果直接单击被加载页面中的链接，Android 系统中的 browser 会响应单击事件，也就是说 Android 会脱离应用程序，切换到系统 browser 显示新页面的内容。如果希望在应用程序中处理单击事件，需要添加 setWebViewClient()方法，代码如下：

```
public class MyWebView extends Activity{
    private WebView mweb;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.main);
        mweb = (WebView) findViewById(R.id.web1);
        mweb.loadUrl("file:///android_asset/index.html");
                                    //加载 assets 文件夹下页面
    }
    mweb.setWebViewClient(new WebViewClient(){
                                    //为 WebView 组件对象添加了一个事件监听器
        public boolean shouldOverrideUrlLoading(WebView view, String url)
        {    //重写方法
```

```
            view.loadUrl(url);              //为 WebView 对象加载新的 url
            return true;
        }
    });
}
```

当使用浏览器浏览互联网页面的时候，可以使用 back 功能返回以前浏览过的页面。在 Android 应用程序中，可以实现类似的功能。这需要重写 Activity 的 onKeyDown 事件，判断当用户按下“返回”按钮，WebView 返回上一页。否则 Activity 会调用自身的结束方法，直接退出当前的 Activity。代码如下：

```
public class MyWebView extends Activity{
    private WebView mweb;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.main);
        mweb = (WebView) findViewById(R.id.web1);
        mweb.loadUrl("file:///android_asset/index.html");
                                    //加载 assets 文件夹下页面
    }
    mweb.setWebViewClient(new WebViewClient(){
                                    //为 WebView 组件对象添加了一个事件监听器
        public boolean shouldOverrideUrlLoading(WebView view, String url)
        {                           //重写方法
            view.loadUrl(url);      //为 WebView 对象加载新的 url
            return true;
        }
    });
    public boolean onKeyDown(int keyCode, KeyEvent event) {
    //KeyEvent.KEYCODE_BACK 常量表示手机上的“返回”按钮
    //canGoBack()方法可以判断是否有页面可以返回
        if ((keyCode == KeyEvent.KEYCODE_BACK) && mweb.canGoBack()) {
            mweb.goBack();          //goBack ()方法返回上一个浏览页面
            return true;
        }
        return super.onKeyDown(keyCode, event);
    }
}
```

11.1.6　对 JavaScript 的支持

作为脚本语言，JavaScript 是页面重要的组成部分，WebView 要支持 JavaScript，需要在代码中加入如下语句：

```
mweb.getSettings().setJavaScriptEnabled(true);//使 WebView 支持 JavaScript
```

WebView 可以很好地支持 JavaScript 绝大部分的功能，稍微有些麻烦的是 JavaScript 中 alert、confirm 等弹出的对话框。在默认情况下，WebView 会忽略掉这些信息框，需要重写 WebView 中 WebChromeClient 对象的一些方法，让 WebView 支持这些操作。先将 assets 文件夹下的 index.html 页面修改一下，代码如下：

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

```
    </head>
    <script language="JavaScript">
        function showAlert(){
            alert("信息窗口弹出成功!");
        }
        function showConfirm(){
            confirm("要关闭当前窗口吗？");
        }
    </script>
    <body>
        这是程序内置测试页面
        <br/>
        <img src="flower.jpg" />
        <br/>
        <a href="javascript:showAlert()">alert 弹出窗口</a>
        <br/>
        <a href="javascript:showConfirm()">confirm 窗口</a>
    </body>
</html>
```

然后修改 Java 代码，添加如下内容：

```
public class MyWebView extends Activity{
    private WebView mweb;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.main);
        mweb = (WebView) findViewById(R.id.web1);
        mweb.loadUrl("file:///android asset/index.html");
                                    //加载 assets 文件夹下页面
    }
    mweb.setWebViewClient(new WebViewClient(){
                                    //为 WebView 组件对象添加了一个事件监听器
        public boolean shouldOverrideUrlLoading(WebView view, String url) {
                                    //重写方法
            view.loadUrl(url);      //为 WebView 对象加载新的 url
            return true;
        }
    });
    public boolean onKeyDown(int keyCode, KeyEvent event) {
    //KeyEvent.KEYCODE BACK 常量表示手机上的“返回”按钮
    //canGoBack()方法可以判断是否有页面可以返回
        if ((keyCode == KeyEvent.KEYCODE BACK) && mweb.canGoBack()) {
            mweb.goBack();          //goBack()方法返回上一个浏览页面
            return true;
        }
        return super.onKeyDown(keyCode, event);
    }

    mweb.getSettings().setJavaScriptEnabled(true);
    mweb.setWebChromeClient(new WebChromeClient(){
        //重写 WebChromeClient 的 onJsAlert()方法支持 JavaScrip 的 alert 信息框
        public boolean onJsAlert(WebView view, String url, String message,
        final JsResult result) {
            Builder builder = new Builder(MyWebView.this);
            builder.setTitle("信息提示");
            builder.setMessage(message);
                                //message 参数就是 JavaScript 中提示框的信息
```

```
            builder.setPositiveButton("确定", new AlertDialog.
            OnClickListener(){  //添加“确认”按钮
                public void onClick(DialogInterface arg0, int arg1) {
                    result.confirm();
                }
            });
            builder.setCancelable(false);//设置信息框可否被手机“返回”按钮撤销
            builder.create();
            builder.show();
            return true;
        }

        //重写 WebChromeClient 的 onJsConfirm()方法支持 JavaScrip 的 alert 信息框
        public boolean onJsConfirm(WebView view, String url,String message,
        final JsResult result) {
            Builder builder = new Builder(MyWebView.this);
            builder.setTitle("信息确认");
            builder.setMessage(message);
                                    //message 参数就是 JavaScript 中提示框的信息
            builder.setPositiveButton("确定", new AlertDialog.
            OnClickListener(){           //添加“确认”按钮
                public void onClick(DialogInterface arg0, int arg1) {
                    result.confirm();
                }
            });
            builder.setNeutralButton ("取消", new AlertDialog.
            OnClickListener(){           //添加“取消”按钮
                public void onClick(DialogInterface arg0, int arg1) {
                    result.cancel();
            }
            });
            builder.setCancelable(false);//设置信息框可否被手机“返回”按钮撤销
            builder.create();
            builder.show();
            return true;
      }
    });
}
```

效果如图 11.3 和图 11.4 所示。



图 11.3　WebView 弹出信息提示框 1



图 11.4　WebView 弹出信息提示框 2

11.2　访问因特网——HTTP 连接

在实际应用中，应用程序有很多情况需要与远端服务器进行信息交互，比如，用户登录信息的验证、向服务器发送数据、获取网络数据等。在本节中，主要来研究一下，如何通过常用的 get 和 post 方式将信息从手机发送到网络服务器上。

11.2.1　准备工作

需要准备一个在服务器上运行的 Web 程序，用来接收终端发来的信息。这个程序很简单，不需要页面，只用一个普通的 Servlet 来接收请求信息。

这个 Servlet 的 doGet()和 doPost()方法中的代码是相同的，都是接收参数并在控制台上输出，以测试手机应用程序是否成功的将信息发送到服务器端，代码如下：

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
    //接收手机终端发送的信息
    String username=request.getParameter("username");
    String password=request.getParameter("password");
    String email=request.getParameter("email");
    //控制台打印信息
    System.out.println("这里是 doGet 方法");
    System.out.println("username: "+username);
    System.out.println("password: "+password);
    System.out.println("email: "+email);
}

public void doPost (HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
    //接收手机终端发送的信息
    String username=request.getParameter("username");
    String password=request.getParameter("password");
    String email=request.getParameter("email");
    //控制台打印信息
    System.out.println("这里是 doPost 方法");
    System.out.println("username: "+username);
    System.out.println("password: "+password);
    System.out.println("email: "+email);
}
```

11.2.2　编写手机端界面文件

手机端的应用程序要向服务器端发送用户名、密码及邮箱等信息，在 res/layout/main.xml 文件中添加如下代码，使用线形布局，将所需的组件元素添加到布局文件中，运行效果如图 11.5 所示。



图 11.5　手机端运行界面

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout height="fill parent"  android:background="#ffffffff">
    <LinearLayout android:orientation="vertical"
        android:layout_width="fill_parent" android:layout_height="wrap_
        content"
        android:layout_marginLeft="25px" android:layout_marginRight="15px"
        android:layout_marginTop="15px">
        <LinearLayout android:orientation="horizontal"
            android:layout_width="wrap_content" android:layout_height="fill_
            parent"
            android:layout marginBottom="15px">
            <TextView android:layout width="90px"
                android:layout height="wrap content" android:text="用户名: "
                android:textSize="17px" android:textColor="#ff000000"
                android:gravity="right"/>
            <EditText android:id="@+id/username" android:layout width="180px"
                android:layout height="wrap content" android:singleLine=
                "true"
                android:hint="输入用户名" android:background="#D6D6D8"
                android:paddingLeft="10px"/>
        </LinearLayout>
        <LinearLayout android:orientation="horizontal"
            android:layout width="wrap content" android:layout height="wrap
            content"
            android:layout marginBottom="15px">
            <TextView android:layout width="90px"
                android:layout_height="wrap_content" android:text="密码 : "
                android:textSize="17px" android:textColor="#ff000000"
                android:gravity="right"/>
            <EditText android:id="@+id/password" android:layout_width="180px"
                android:layout_height="wrap_content" android:password="true"
                android:hint="输入密码" android:singleLine="true" android:
                background="#D6D6D8"
                android:paddingLeft="10px"/>
```

```
        </LinearLayout>
        <LinearLayout android:orientation="horizontal"
            android:layout_width="wrap_content" android:layout_height=
            "wrap_content">
            <TextView android:layout_width="90px"
                android:layout_height="wrap_content" android:text="邮箱 ：  "
                android:textSize="17px" android:textColor="#ff000000"
                android:gravity="right"/>
            <EditText android:id="@+id/email" android:layout_width="180px"
                android:layout_height="wrap_content"
                android:hint="输入邮箱" android:singleLine="true" android:
                background="#D6D6D8"
                android:paddingLeft="10px"/>
        </LinearLayout>
    </LinearLayout>
    <Button android:id="@+id/login" android:layout_width="113px"
        android:layout_height="wrap_content" android:text="发   送"
        android:layout_marginLeft="111px" android:layout_marginTop="15px"
        android:textSize="17px"
        />
</LinearLayout>
```

11.2.3　发送 get 请求

与其他类型的应用程序一样，Android 应用程序发送 HTTP 请求也是遵循以下步骤：

（1）获取请求的 URL 地址。

（2）创建 HTTP 连接及请求所需对象。

（3）设置连接参数。

（4）选择请求类型执行操作。

（5）判断服务器端响应状态。

（6）处理返回结果。

Android 在 1.5 版本以后，整合了开源的 Apache HttpClient 项目，可以使开发者高效的解决以上问题。实现对网站的 HTTP GET 请求，代码如下：

```
public class ConnectWeb {
    public boolean sendGetRequest(String username,String password,String
    email){
        boolean pan=false;
        try{
            //准备请求的 URL 地址，所需参数附加在 URL 上
            String url="http://192.168.1.8:8080/AndroidWeb/MyServlet?
            username="+username+"&password="+password+"&email="+email;
            HttpGet request = new HttpGet(url);    //实例化 HttpGet 对象
            //使用 Apache 的缺省实现 DefaultHttpClient 实例化 HttpClient 对象
            HttpClient httpClient = new DefaultHttpClient();
            HttpResponse response = httpClient.execute(request);
                                        //HttpClient 对象执行 execute 操作
            //判断 HttpResponse 对象返回的状态码，当网站正常接收到请求信息并返回结
              果的时候
            //返回码的数值是 200，可以由 HttpStatus 对象的常量 SC_OK 表示
            if(response.getStatusLine().getStatusCode()==HttpStatus.SC_OK){
                pan=true;
            }
```

```
        }
        catch(Exception e){
            e.printStackTrace();
        }
        return pan;
    }
}
```

以上代码中，也可以为 DefaultHttpClient 添加 HttpParams 类型的参数。

在界面文件中调用该方法，运行程序，效果如图 11.6 所示。同时部署在服务器上的 Web 程序，会在控制台上打印出信息，效果如图 11.7 所示。



图 11.6　发送 get 请求

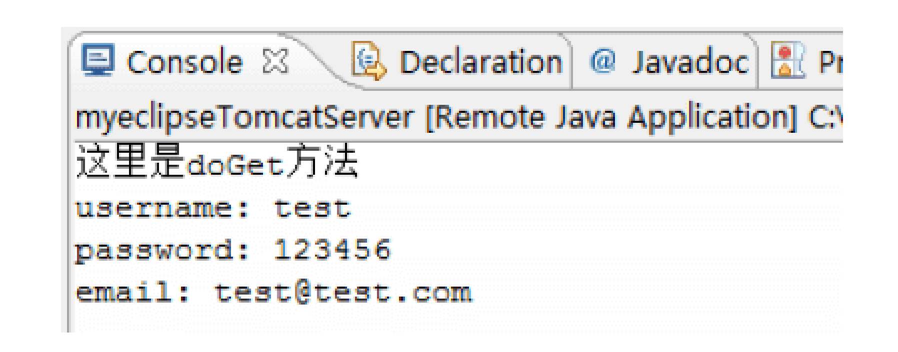


图 11.7　get 请求效果

11.2.4　发送 post 请求

与 get 请求相比，post 请求在流程上没有变化，区别在于使用的请求对象类型不同，还需要专门的对象来为 post 请求的参数进行设置，代码如下：

```
public class ConnectWeb {
    public boolean sendGetRequest(String username,String password,String
    email){
        boolean pan=false;
        try{
            //准备请求的 URL 地址，所需参数附加在 URL 上
            String url="http://192.168.1.8:8080/AndroidWeb/MyServlet?
            username="+username+"&password="+password+"&email="+email;
            HttpGet request = new HttpGet(url);     //实例化 HttpGet 对象
            //使用 Apache 的默认实现 DefaultHttpClient 实例化 HttpClient 对象
            HttpClient httpClient = new DefaultHttpClient();
            HttpResponse response = httpClient.execute(request);
                                        //HttpClient 对象执行 execute 操作
            //判断 HttpResponse 对象返回的状态码，当网站正常接收到请求信息并返回结
              果的时候
            //返回码的数值是 200，可以由 HttpStatus 对象的常量 SC_OK 表示
            if(response.getStatusLine().getStatusCode()==HttpStatus.SC_OK){
                pan=true;
            }
```

```
        }
        catch(Exception e){
            e.printStackTrace();
        }
        return pan;
    }

    public boolean sendPostRequest(String username,String password,String
    email){
        boolean pan=false;
        try{
            String url="http://192.168.1.8:8080/AndroidWeb/MyServlet";
                                                    //请求的 URL 地址
            HttpPost request = new HttpPost(url);  //实例化 HttpPost 对象
            //post 请求传递参数，需要放在 NameValuePair 的集合中
            List <NameValuePair> params = new ArrayList <NameValuePair>();
            params.add(new BasicNameValuePair("username", username));
                                                    //放置 username 参数
            params.add(new BasicNameValuePair("password", password));
                                                    //放置 password 参数
            params.add(new BasicNameValuePair("email", email));
                                                    //放置 email 参数
            request.setEntity(new UrlEncodedFormEntity(params,
            HTTP.UTF_8));                           //发出 post 请求
            HttpResponse response = new DefaultHttpClient().execute(request);
            //判断 HttpResponse 返回状态
            if(response.getStatusLine().getStatusCode()==HttpStatus.SC_OK){
                pan=true;
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
        return pan;
    }
}
```

程序运行效果如图 11.8 所示。同时部署在服务器上的 Web 程序，也会在控制台上打印出信息，效果如图 11.9 所示。



图 11.8　发送 post 请求

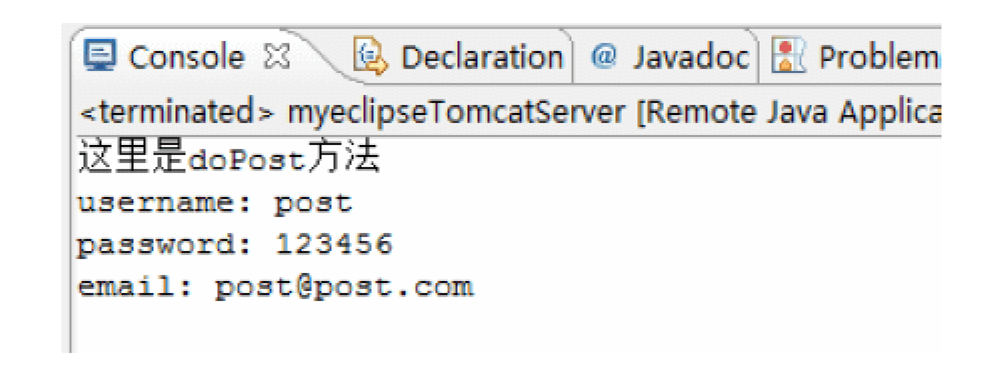


图 11.9　post 请求效果

11.3　解析服务器端返回的 XML 数据

在上一节中，已经成功地把手机端的信息发送到了服务器端，通常情况下，服务器端会返回一些数据。在本节中，就来研究如何解析服务器端返回的数据信息。

HttpResponse 对象会将服务器端返回的数据封装在一个 HttpEntity 对象中，最直接的方法是获取这个 HttpEntity 对象的字符串形式，然后按照某种格式来解析这个字符串。常见的格式有 XML 和 JSON 两种，先来看一下 Android 如何解析 XML 格式数据。

11.3.1　准备工作

首先，需要在 Web 服务器端的程序中添加一个 Servlet，这个 Servlet 被请求后返回如下格式的 XML 数据：

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
    <user>
        <name>张三</name>
        <phone>123456789</ phone >
    </user>
    <user>
        <name>李四</name>
        <phone>987654321</ phone >
    </user>
</result>
```

然后在手机端准备一个 POJO 对象，用来封装 XML 所包含的信息，代码如下：

```
public class UserBean {
    private String name;
    private String phone;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
}
```

接下来，要编写手机端的请求方法，来获取这个 XML 数据，ConnectWeb.java 代码如下：

```
public class ConnectWeb {
    public List<UserBean> getUserList(){
        List<UserBean> ulist=new ArrayList<UserBean>();
        try{
            String  url="http://192.168.1.8:8080/AndroidWeb/XMLServlet";
                                                       //获取请求的 URL
```

```
            HttpPost request = new HttpPost(url);  //实例化 HttpPost 对象
            HttpResponse response = new DefaultHttpClient().
            execute(request);                      //发出请求
            if(response.getStatusLine().getStatusCode()==HttpStatus.SC OK){
                                                   //判断返回码
                //将从服务器端获取的数据转换成字符串形式
                String str = EntityUtils.toString(response.getEntity());
                ulist=new ParseDOM().parseByDOM(str);
                                           //准备以 DOM 方式解析(在下一节介绍)
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
        return ulist;
    }
}
```

11.3.2　以 DOM 方式解析数据

作为一个基于 Java 语言的平台，Android SDK 支持 JRE 的大部分功能，这其中就包括对 DOM 和 SAX 的支持。

DOM 方式以树状模型方式解析 XML 数据，这有助于开发者写出非常直观的代码，在某些功能方面比 SAX 解析方法简单。但是这种方式需要将全部 XML 文档读进内存中，内存占用对移动设备来说是一个头疼的问题，所以建议只在解析少量数据时采用。下面开始以 DOM 方式解析数据，代码如下：

```
public class ParseDOM {
    public List<UserBean> parseByDOM(String str){
        List<UserBean> ulist=new ArrayList<UserBean>();//准备 users 对象集合
        try{
            //获取 dom 解析器工厂实例
            DocumentBuilderFactory domfac=DocumentBuilderFactory.newInstance();
            DocumentBuilder dombuilder=domfac.newDocumentBuilder();
                                                   //获取 dom 解析器
            InputStream is=new ByteArrayInputStream(str.getBytes());
                                                   //将解析数据转化为输入流
            Document doc=dombuilder.parse(is);      //解析获得 dom 对象
            Element root=doc.getDocumentElement(); //获取 dom 树的根节点
            NodeList users=root.getChildNodes();   //获取子节点<users>列表
            for(int i=0;i<users.getLength();i+=1){
                Node u=users.item(i);              //获取单个<users>子节点对象
                NodeList p=u.getChildNodes();      //获取<users>节点下元素列表
                if(p.getLength()>0){
                    UserBean ub=new UserBean();//准备 users 对象
                    for(int j=0;j<p.getLength();j+=1){
                    Node atribue=p.item(j);
                        if(atribue.getNodeName().equals("name")){
                            ub.setName(atribue.getFirstChild().getNode-
                            Value());              //为 users 对象赋值
                        }
                        else if(atribue.getNodeName().equals("phone")){
```

```
                        ub.setPhone(atribue.getFirstChild().getNode
                        Value());           //为 users 对象赋值
                    }
                }
                ulist.add(ub);              //添加 users 对象到集合中去
            }
        }
    }
    catch(Exception e){
        e.printStackTrace();
    }
    return ulist;
  }
}
```

将解析完成后得到的对象列表返回给显示界面，运行效果如图 11.10 所示。

11.3.3　以 SAX 方式解析数据

与 DOM 方式不同，SAX 在解析 XML 文档时采用了基于事件的模型。它在解析 XML 数据会触发一系列的事件，查找指定的标记，并在相应的回调方法，获取标记中的值。

SAX 解析方式对内存的要求通常会比较低，它不需要把全部数据读进内存。它可以让开发人员自己来决定所要处理的标签，特别是当开发人员只需要处理文档中所包含的部分数据时，SAX 这种扩展能力得到了更好的体现。但是 SAX 解析器的编码会比较复杂，而且很难同时访问同一个文档中的多处不同数据。



图 11.10　DOM 解析结果

在 Android 应用程序中，同样可以直接继承 DefaultHandler 类来进行开发。使用 DefaultHandler 类方式解析数据的代码如下：

```
public class ParseSAX extends DefaultHandler{
    private List<UserBean> ulist;
    private UserBean currentUser;
    private String temp;
    public List<UserBean> getUsers() {
        return this.ulist;
    }

    public void startDocument() throws SAXException {  //文档解析开始时触发
        ulist = new ArrayList<UserBean>();              //准备返回的对象集合
    }

    public void startElement(String uri, String localName, String name,
                                                        //解析元素标签时触发
            Attributes attributes) throws SAXException {
        if (name.equalsIgnoreCase("user")) {    //判断是否处理到<user>标签
            this.currentUser = new UserBean(); //实例化一个 UserBean 对象
        }
    }
```

```
    public void characters(char[] ch, int start, int length)
                                              //解析到元素标签的内容时触发
            throws SAXException {
        temp=new String(ch,start,length);     //获取标签内容
    }

    public void endElement(String uri, String localName, String name)
                                              //标签解析完毕时触发
            throws SAXException {
        if (this.currentUser != null) {
            if (name.equalsIgnoreCase("name")) {//判断是否解析完毕<name>标签
                currentUser.setName(temp); //为 UserBean 对象 name 属性赋值
            } else if (name.equalsIgnoreCase("phone")) {
                                          //判断是否解析完毕<phone>标签
                currentUser.setPhone(temp);//为 UserBean 对象 phone 属性赋值
            } else if (name.equalsIgnoreCase("user")) {
                                          //判断是否解析完毕< user >标签
                ulist.add(currentUser);   //将 UserBean 对象加入集合
            }
        }
    }
}
```

上面的代码是为解析服务器端的数据准备的解析代码，需要在适当的时候调用它完成解析工作，代码如下：

```
public class ConnectWeb {
    public List<UserBean> getUserList(){
        List<UserBean> ulist=new ArrayList<UserBean>();
        try{
            String  url="http://192.168.1.8:8080/AndroidWeb/XMLServlet";
                                                  //获取请求的 URL
            HttpPost request = new HttpPost(url); //实例化 HttpPost 对象
            HttpResponse response = new DefaultHttpClient().execute
            (request);                            //发出请求
            if(response.getStatusLine().getStatusCode()==
            HttpStatus.SC_OK){                    //判断返回码
                //将从服务器端获取的数据转换成字符串形式
                String str = EntityUtils.toString(response.getEntity());
                ulist= sax (str);                 //准备以 SAX 方式解析
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
        return ulist;
    }

    public List<UserBean> sax(String str){
        List<UserBean> ulist=new ArrayList<UserBean>();
        try{
            SAXParserFactory factory = SAXParserFactory.newInstance();
                                                      //获取解析工厂实例
            SAXParser parser = factory.newSAXParser(); //获取解析器实例
            ParseSAX handler = new ParseSAX();        //实例化自定义解析工具
            parser.parse(new ByteArrayInputStream(str.getBytes()), handler);
            ulist=handler.getUsers();
```

```
        }
        catch(Exception e){
            e.printStackTrace();
        }
        return ulist;
    }
}
```

代码说明：

- DefaultHandler 类继承了 ContentHandler、DTDHandler、EntityResolver、ErrorHandler 的所有接口。
- 一般在 startDocument 中进行初始化工作，在 endDocument 处理收尾工作。
- 一个 XML 节点读取的过程按照 startElement、characters、endDocument 顺序进行。

11.3.4　Android 基于 SAX 的解析器解析数据

SAX 方式的优点十分明显，但是其不宜编写程序的缺陷也是很让人头疼的，因此出现了不少基于 SAX 解析器的解析方式，Android SDK 也提供了自己独有的 SAX API。它并未使用 SAX 处理程序，而是使用了 SDK 中的 android.sax 包中的类。这些类允许您构建 XML 文档的结构，并根据需要添加事件监听程序。用 Android 的 SAX API 解析数据，代码如下：

```
public class ParseAndroid {
    public List<UserBean> parse(String str) {
        final List<UserBean> ulist = new ArrayList<UserBean>();
                                        //准备返回的UserBean对象集合
        final UserBean currentUser = new UserBean();
                                        //准备当前操作的UserBean对象

        RootElement root = new RootElement("result");  //准备数据解析根元素
        Element item = root.getChild("user");          //准备数据节点元素
        item.getChild("name").setEndTextElementListener(
                                        //当<name>标记结束后触发
                new EndTextElementListener() {
                                  //设置 EndTextElementListener 监听器
                    public void end(String body) {
                        currentUser.setName(body);//为 UserBean 对象属性赋值
                    }
                });
        item.getChild("phone").setEndTextElementListener(
                                  //当<phone>标记结束后触发
                new EndTextElementListener() {
                                  //设置 EndTextElementListener 监听器
                    public void end(String body) {
                        currentUser.setPhone(body);
                                  //为 UserBean 对象属性赋值
                    }
                });
        item.setEndElementListener(new EndElementListener() {
                                  //当<user>标记结束后触发
            public void end() {           //设置 EndTextElementListener 监听器
                ulist.add(currentUser.copy()); //将 UserBean 对象添加进集合
            }
        });
```

```
        try {
            Xml.parse(new ByteArrayInputStream(str.getBytes()),
                    Xml.Encoding.UTF_8, root.getContentHandler());
                                                    //开始解析 XML 数据
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        return ulist;
    }
}
```

代码说明：

- 代码中声明文档将有一个 result 根元素节点，并且它有多个 user 子元素节点。
- 然后，为 user 元素的多个字元素节点添加监听程序。
- 最后，调用 Xml.parse()方法，传递一个通过根元素生成的处理程序为参数。

11.3.5　Android XML PULL 解析器

同样以流的方式解析 XML 数据，与 SAX 将事件“推入”处理程序不同，StAX 是以“拉取”的方式处理 XML 数据中的节点。Android 并未提供对 Java StAX API 的支持，但是，Android 附带了一个 pull 解析器，其工作方式类似于 StAX。Android XML PULL 解析器解析数据的示例代码如下：

```
public class AndroidPull {
    public List<UserBean> parse(String str) {
        final List<UserBean> ulist = new ArrayList<UserBean>();
        XmlPullParser parser = Xml.newPullParser();//实例化 pull 解析器对象
        try{
            //以 utf-8 编码设置解析器准备解析的文档
            parser.setInput(new ByteArrayInputStream(str.getBytes()), "utf-8");
            //获取解析器当前事件，如文档开始事件、标签开始事件等
            int eventType = parser.getEventType();
            UserBean user=null;                 //定义当前被解析的 user 对象
            while (eventType != XmlPullParser.END_DOCUMENT ){
                                                //设置当文档结束时解析结束
                String temp = null;
                switch (eventType){
                case XmlPullParser.START_TAG:   //当标签开始事件发生时触发
                    temp=parser.getName();      //获取当前触发事件的标签名称
                    if(temp.equalsIgnoreCase("user")){
                                            //当 user 标签开始时实例化 user 对象
                        user=new UserBean();
                    }
                    //当 name 标签开始时为 user 对象的 name 属性赋值
                    else if(temp.equalsIgnoreCase("name")){
                        user.setName(parser.nextText());
                    }
                    //当 phone 标签开始时为 user 对象的 phone 属性赋值
                    else if(temp.equalsIgnoreCase("phone")){
                        user.setPhone(parser.nextText());
                    }
                    break;
                case XmlPullParser.END_TAG:     //当标签结束事件发生时触发
                    temp=parser.getName();      //获取当前触发事件的标签名称
```

```
                    if(temp.equalsIgnoreCase("user")){
                                        //当 user 标签结束时将 user 对象放入集合
                        ulist.add(user);
                    }
                    break;
                }
                eventType = parser.next();      //开始解析下一个标签
            }

        }
        catch(Exception e){
            e.printStackTrace();
        }
        return ulist;
    }
}
```

代码说明：

- pull 解析器提供多种事件，如开始元素和结束元素等，需要使用 parser.next()方法提取它们。
- pull 解析器的事件将作为数值代码被发送。
- 当某个元素开始被处理时，可以调用 parser.nextText()方法从 XML 文档中提取所有数据。

11.4　解析服务器端返回的 JSON 数据

尽管 XML 具备跨平台、可扩展等多种优势，但是为了解析 XML 格式数据的复杂代码，带来了开发效率的低下。除非是 Web Services 应用，否则在大多数 Web 应用中，开发者根本不需要使用复杂的 XML 格式来传递数据。

在这种情况下，JSON 作为一种轻量级的数据交换格式出现了。同 XML 相比，它结构简单操作灵活，易于人阅读和编写，同时也易于机器解析和生成。更重要的是，由于 JSON 不使用需要匹配的标签，大大降低了传送信息的字节数。

11.4.1　准备工作

同上一节的准备工作相同，也要在 Web 服务器端的程序准备一个 Servlet，用来返回 JSON 格式的数据，代码如下：

```
[
    {"city":"北京"," postcode":"100000"},
    {"city":"上海","postcode":"200001"}
]
```

代码说明：

- 服务器段返回的是由两个对象组成的对象数组。
- 在 JSON 结构中，数组由一对中括号包围，数组中多个对象之间由逗号分隔。
- 每个数据对象由一对大括号包围，对象中多个属性由逗号分隔。

❑ 每个对象属性由一对 key、value 组成，中间由冒号分隔。

在手机端获取 JSON 数据的代码同获取 XML 数据的代码是一样的，只需要准备一段解析 JSON 数据结构的代码即可。同样也要在手机端准备一个 POJO，用来封装 city 对象，代码如下：

```
public class CityBean {
    private String name;
    private String code;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getCode() {
        return code;
    }
    public void setCode(String code) {
        this.code = code;
    }
}
```

11.4.2　解析 JSON 数据

在 JSON 的官方网站上可以看到，目前支持 JSON 格式的语言越来越多。作为主流编程语言的 Java 当然不会例外。在 Android 应用程序中，可以使用 org.json 包中的对象快速简便地完成对 JSON 数据的解析，代码如下：

```
public class ConnectWeb {
    public List<CityBean> getCityList(){
        List<CityBean> clist=null;
        try{
            String url="http://192.168.1.8:8080/AndroidWeb/JSONServlet";
                                                  //获取请求的 URL
            HttpPost request = new HttpPost(url); //实例化 HttpPost 对象
            HttpResponse response = new DefaultHttpClient().execute
            (request);                            //发出请求
            if(response.getStatusLine().getStatusCode()==
            HttpStatus.SC_OK){                    //判断返回码
                //将从服务器端获取的数据转换成字符串形式
                String str = EntityUtils.toString(response.getEntity());
                clist=getCList(str);              //准备以 JSON 方式解析
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
        return clist;
    }

    private List<CityBean> getCList(String str){
        List<CityBean> clist=new ArrayList<CityBean>();
        try{
            JSONArray jay=new JSONArray(str);   //将字符串信息转化成 JSON 数组
```

```
            for(int i=0;i<jay.length();i+=1){
                JSONObject temp=(JSONObject)jay.get(i);
                                        //将数组中的每个对象转化成为 JSON 对象
                CityBean city=new CityBean();       //实例化 city 对象
                city.setName(temp.getString("city"));
                                        //获取 JSON 对象数值为 city 属性复制
                city.setCode(temp.getString("postcode"));
                                        //获取 JSON 对象数值为 city 属性复制
                clist.add(city);        //将 city 对象添加到集合
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
        return clist;
    }
}
```

代码说明：

- Java 解析 JSON 数据经常用到 JSONObject 和 JSONArray 对象，这两个对象都可以通过字符串直接实例化，前提是该字符串符合 JSON 数据格式。
- 可以向操作普通数组一样遍历或根据下标查找 JSONArray 中的元素。
- JSONArray 数组中的元素可以使字符串、数字，更普遍的是 JSONObject 对象。
- 可以很方便地以 key/value 的方式获取 JSONObject 对象中的各个属性。

上例中程序的运行结果如图 11.11 所示。



图 11.11　JSON 解析结果

到目前为止，介绍了 Android 应用程序与服务器端的程序进行交互的一些基本方式，看起来似乎与普通的客户端程序没什么两样。但是在实际的开发过程中，总是会有一些其他的需求出现，从下一节开始，将会介绍 Android 应用程序与互联网交互的其他方式。

11.5　获取网络资源——HttpURLConnection

11.5.1　显示网络图片

如果应用程序希望显示一张互联网上的图片，而又不想把该图片下载下来存储在手机上，那么 HttpURLConnection 对象正符合这样的需求。

HttpURLConnection 对象可以打开给定的 HTTP 网址，并将数据以字符流的形式获取到手机端，然后就可以根据需要来处理这些数据。

下面是一个使用 HttpURLConnection 对象显示网络图片的例子，首先要根据一个网址来获取要显示的图片的字符流，代码如下：

```
public class ConnectWeb {
    public Bitmap getPicBitmap(){
        Bitmap bitmap = null;                  //声明 Bitmap 对象
        try{
           String url="http://192.168.1.8:8080/AndroidWeb/pics/flower.jsp";
                                               //要显示的图片地址
            URL picUrl = new URL(url);         //根据地址实例化 URL 对象
                                        //使用 URL 对象生成 HttpURLConnection 对象
            HttpURLConnection conn = (HttpURLConnection) picUrl.open
            Connection();
            conn.connect();             //使用 HttpURLConnection 对象打开连接
            if(conn.getResponseCode()==200){   //判断是否正常得到结果
                InputStream ins = conn.getInputStream();
                                        //以 InputStream 形式取得服务器端的数据
                bitmap = BitmapFactory.decodeStream(ins);
                                        //转化数据实例化 bitmap 对象
                ins.close();                   //关闭数据流
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
        return bitmap;
    }
}
```

然后，就需要准备一个显示图片的界面，代码如下：

```
public class ShowPic extends Activity{
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.main);    //加载样式 xml
        ImageView imageView = (ImageView) findViewById(R.id.imageView);
                                               //实例化 ImageView 对象
        imageView.setImageBitmap(new ConnectWeb().getPicBitmap());
                                               //显示图片
    }
}
```

最终的运行效果，如图 11.12 所示。

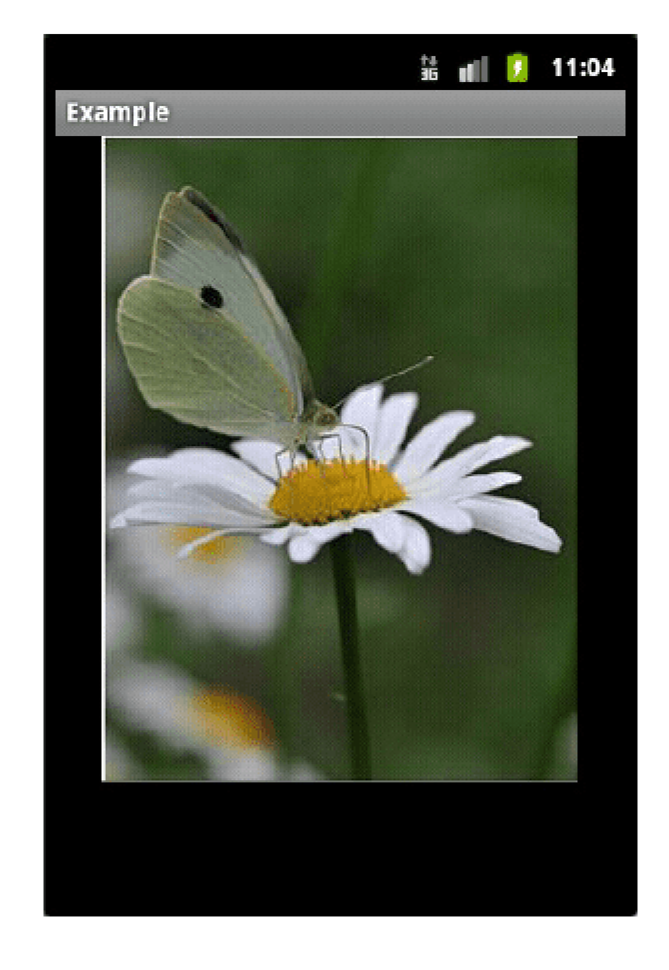


图 11.12　显示网络图片

11.5.2　下载网络音乐

HttpURLConnection 对象以字符流的形式获取数据，同样可以利用它来实现对网络音乐的下载。本节中要来实现这样一个功能：从网络上下载一个 MP3 到手机上，然后播放它。

MP3 文件会被下载到手机的 SD 卡上，下载一个 MP3 需要的时间比较长，所以在界面要准备一个下载进度条，当下载完成后，进度条消失，再使用 MediaPlayer 来播放它。

首先，准备下载的代码，仍然使用 HttpURLConnection 对象，代码如下：

```
public class ConnectWeb {
    public HttpURLConnection getMP3Stream(){
        HttpURLConnection conn = null;
        try{
            //准备下载的 MP3 文件地址
            String url="http://192.168.1.8:8080/AndroidWeb/mp3/hasta_
            siempre_comandante.mp3";
            URL picUrl = new URL(url);                  //根据地址实例化 URL 对象
            //生成 HttpURLConnection 连接对象
            HttpURLConnection conn = (HttpURLConnection) picUrl.openConnection();
            conn.connect();                              //打开连接
            if(conn.getResponseCode()!=200){             //判断连接状态
                conn = null;    //返回状态不正确将 HttpURLConnection 置为空值
            }
        }
        catch(Exception e){
            e.printStackTrace();
            conn = null;
        }
        return conn;
    }
}
```

代码说明：

在这个连接中，没有返回 HttpURLConnection 对象获得的 InputStream，而是返回了 HttpURLConnection 本身，因为在界面设计上需要它的帮助。

现在，来准备界面，界面中的控件包括一个下载按钮和对话框，代码如下：

```
public class DownMP3  extends Activity{
    private Button button;
    private ProgressDialog dDialog;
    private int fileSize = 0;
    private int dowloaded = 0;
    private byte[] bytes;
    private MediaPlayer player = new MediaPlayer();
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.mp3layout);        //加载布局文件

        button = (Button) findViewById(R.id.button);    //实例化按钮控件
        button.setOnClickListener(new OnClickListener(){//为按钮加监听器
            public void onClick(View v) {
                downMp3();                               //开始下载操作
            }
```

```
        });
    }

    private void downMp3(){

        final HttpURLConnection conn=new ConnectWeb().getMP3Stream();
                                                    //获取连接对象
        if(conn==null){                             //判断连接是否异常
            return;
        }

        dDialog = new ProgressDialog(DownMP3.this);    //实例化下载进度框
        dDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
                                                    //设置进度框风格
        dDialog.setTitle("请稍等...");               //设置进度框标题
        dDialog.setMessage("正在下载中...");          //设置进度框信息
        dDialog.setIndeterminate(false);            //设置进度条模式
        dDialog.setCancelable(false);             //设置进度是否可被取消
        dDialog.show();                           //显示进度框

        fileSize=conn.getContentLength();         //获取下载文件的总长度
        if (fileSize - dowloaded > 2048) {
            bytes = new byte[2048];               //设置下载缓存区域大小
        } else {
            bytes = new byte[fileSize - dowloaded];//设置下载缓存区域大小
        }
        dDialog.setMax(fileSize);                 //设置进度条最大数值
        dDialog.setProgress(0);                   //设置进度框当前进度数值

        new Thread() {                            //启动一个新线程用于下载
            public void run() {
                try {
                    int len = 0;                  //下载计数
                    InputStream iStream = conn.getInputStream();
                                                //以 InputStream 获取下载文件
                    //在 SD 卡上创建一个文件保存下载的文件
                    File file = new File(Environment.getExternalStorage-
                    Directory().getPath()+File.separator+ "mp3"+ File.
                    separator+"1.mp3");
                    if(!file.getParentFile().exists()){
                                                //判断保存文件的路径是否存在
                        file.getParentFile().mkdirs(); //创建路径
                    }
                    OutputStream oStream = new FileOutputStream(file);
                                                //打开写文件的输出流

                    while ((len = iStream.read(bytes)) != -1) {
                        dowloaded += len;       //已下载总长度累加
                        if (dowloaded != fileSize) {//判断是否全部下载完毕
                            dDialog.setProgress(dowloaded);
                                                //更新下载对话框上当前进度数值
                            Thread.sleep(100);  //线程休眠 0.1 秒
                        } else {
                            dDialog.cancel();   //关闭下载对话框
                        }
                        oStream.write(bytes, 0, len);//将缓存区域内容写进文件
```

```
                    }

                    dDialog.cancel();                  //关闭下载对话框
                    oStream.flush();                   //刷新输出流
                    oStream.close();                   //关闭输出流
                    iStream.close();                   //关闭输入流

                    Message m = new Message();         //实例化消息对象
                    mp3Handler.sendMessage(m);         //向 Handler 发出消息
                }
                catch(Exception e){
                    e.printStackTrace();
                    dDialog.cancel();
                }
            }
        }.start();
    }

    Handler mp3Handler = new Handler() {               //下载完成后的 Handler
        public void handleMessage(Message msg) {
            try {
            //为 MediaPlayer 设置播放资源路径
            player.setDataSource(Environment.getExternalStorageDirectory().
            getPath()+"/mp3/1.mp3");
                player.reset();                        //重置 MediaPlayer
                player.prepare();                      //准备 MediaPlayer
                player.start();                        //MediaPlayer 播放
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    };

    public boolean onKeyDown(int keyCode, KeyEvent event) {
                                                       //手机按键事件
        if(keyCode==4 && player.isPlaying()){
                                    //判断是否按下返回及 MediaPlayer 是否在播放中
            player.reset();                            //重置 MediaPlayer
        }
        return super.onKeyDown(keyCode, event);
    }
}
```

代码说明：

- 代码中要在文件下载的同时，更新下载对话框上的数值，所以将下载操作放在一个新的线程中去执行。
- 下载线程每隔一段时间休眠一次，执行更新对话框数值的操作。
- 当用户下载完毕后直接播放下载的 MP3 文件。
- 当用户离开时，要关闭还未播放完毕的 MP3 文件，所以需要监听手机上的“返回”按键。

程序运行效果如图 11.13 所示。



图 11.13　下载网络 MP3

11.6　上传文件到网络服务器

文件上传的功能是网络应用中很常见的一项功能。Android 应用程序同样可以将手机上的文件上传到服务器端，本节就来看一下如何实现这一目标。

11.6.1　准备工作

要再次修改服务器端的程序，添加一个可以接收上传文件的 Servlet，这是一个很常见的应用，有很多集成的框架或开源的项目都提供对文件上传功能的支持。这里采用的是 Apache 的 commons.fileupload，FileUploadServlet.java 代码如下：

```
public class FileUploadServlet extends HttpServlet {
    private String uploadPath ;              //文件上传后的保存位置
    private String tempPath ;                //文件上传过程中的临时位置
    File tempPathFile;                       //临时文件
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {
        //获取工程的实际路径下的上传文件夹位置
        uploadPath=request.getSession().getServletContext().getRealPath("/")
        +"upload";
        tempPath=uploadPath+"/buffer"; //工程实际路径下的上传临时文件夹路径
        try {
            DiskFileItemFactory factory = new DiskFileItemFactory();
                                //创建 DiskFileItemFactory 工厂实例
            factory.setSizeThreshold(4096);            //设置缓冲文件大小
            factory.setRepository(tempPathFile);       //设置缓冲文件路径
            ServletFileUpload upload = new ServletFileUpload(factory);
                                //实例化 ServletFileUpload 对象
            upload.setSizeMax(4194304);                //设置上传文件大小
            List<FileItem> items = upload.parseRequest(request);
                                               //获取上传的文件
            Iterator<FileItem> i = items.iterator();
```

```
            while (i.hasNext()) {                  //遍历所有上传文件
                FileItem fi = (FileItem) i.next(); //获取单个上传文件
                String fileName = fi.getName();    //获取文件名
                if (fileName != null) {
                    File fullFile = new File(fi.getName());//实例化 File 对象
                    File savedFile = new File(uploadPath, fullFile.
                    getName());                    //实例化保存位置的文件
                    fi.write(savedFile);           //写入保存位置文件
                }
            }
            out.println("upload success");         //返回上传成功结果
        } catch (Exception e) {
            e.printStackTrace();
            out.println("upload fail");            //返回上传失败结果
        }
    }

    //其余部分代码省略
    …
}
```

下面，需要准备一个上传的图片。在模拟器的 SD 卡上创建一个目录，然后导入一张图片，作为上传的资源文件，效果如图 11.14 所示。

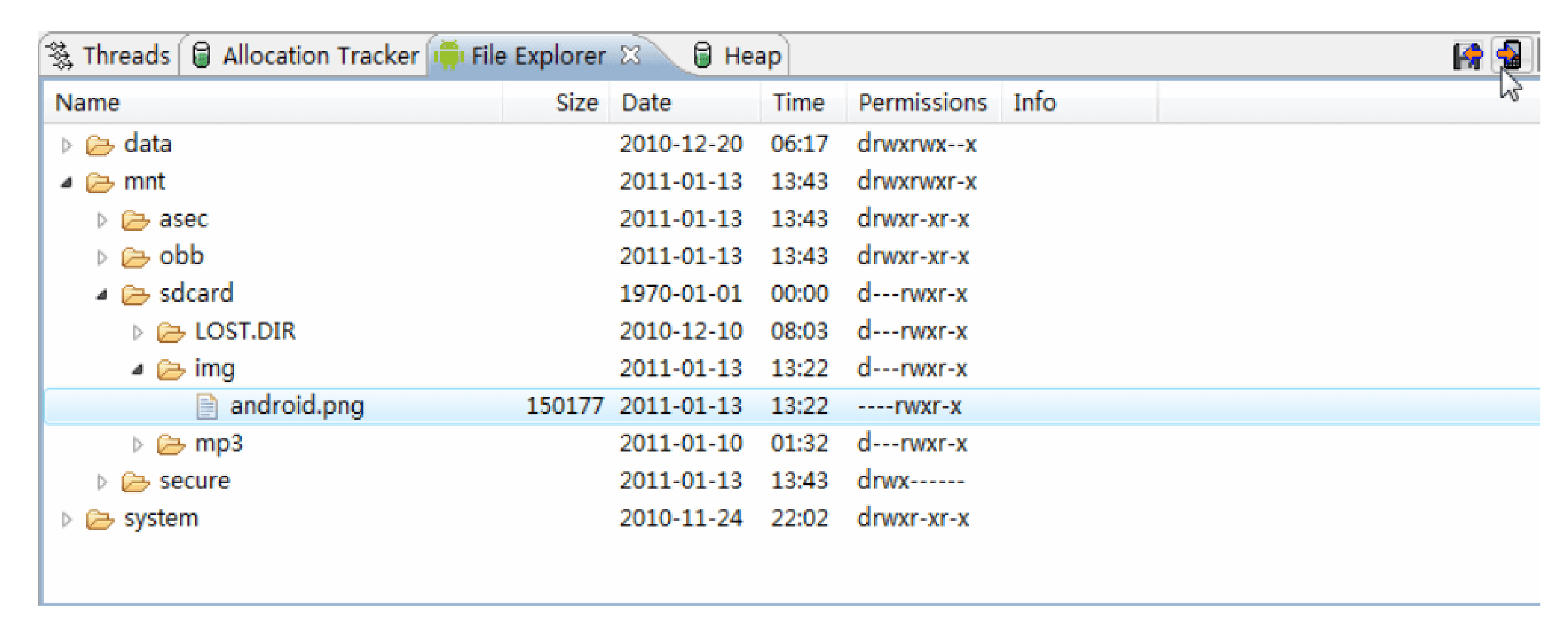


图 11.14　准备上传文件

11.6.2　文件上传代码编写

要准备一个上传文件的方法，这个方法会以字符串的形式返回服务器端的信息，代码如下：

```
public class ConnectWeb {
    //服务器接受上传文件的位置
    private String webUrl = "http://192.168.1.8:8080/AndroidWeb/
    FileUploadServlet";
    //获取手机 SD 卡上文件的位置
    private String filePath = Environment.getExternalStorageDirectory()
            .getPath()+ "/img/android.png";
    private String fileName = "android.png";    //上传文件的名字

    public String uploadFile() {
        String str = "";                         //方法返回值
```

```
        String end = "\r\n";
        String twoHyphens = "--";
        String boundary = "*****";
        try {
            URL url = new URL(webUrl);          //创建 URL 对象
            //实例化 HttpURLConnection 对象
            HttpURLConnection con = (HttpURLConnection) url.openConnection();
            con.setDoInput(true);               //允许使用输入流
            con.setDoOutput(true);              //允许使用输出流
            con.setUseCaches(false);            //不允许使用缓存
            con.setRequestMethod("POST");       //设置以 post 请求方式传输数据
            //设置请求属性
            con.setRequestProperty("Connection", "Keep-Alive");
            con.setRequestProperty("Charset", "UTF-8");
            con.setRequestProperty("Content-Type",
                    "multipart/form-data;boundary=" + boundary);
            //通过 HttpURLConnection 对象打开输出流
            DataOutputStream ds = new DataOutputStream(con.getOutputStream());
            //设置请求头格式
            ds.writeBytes(twoHyphens + boundary + end);
            ds.writeBytes("Content-Disposition: form-data; "
                    + "name=\"file1\";filename=\"" + fileName + "\"" + end);
            ds.writeBytes(end);
            //获取上传文件的输入流
            FileInputStream fStream = new FileInputStream(filePath);
            int bufferSize = 1024;              //设置输出缓存大小
            byte[] buffer = new byte[bufferSize];

            int length = -1;
            while ((length = fStream.read(buffer)) != -1) {
                                                //从文件读取数据至缓存区
                ds.write(buffer, 0, length);    //数据写入/输出流
            }
            ds.writeBytes(end);
            ds.writeBytes(twoHyphens + boundary + twoHyphens + end);
            fStream.close();                    //关闭输入流
            ds.flush();                         //刷新输出流

            InputStream is = con.getInputStream();//取得服务器端返回的信息
            int ch;
            StringBuffer b = new StringBuffer();
            while ((ch = is.read()) != -1) {
                                //将返回信息写入 StringBuffer 对象
                b.append((char) ch);
            }
            str = b.toString().trim();
                                //将 StringBuffer 对象转化为 String 对象返回
            ds.close();
        } catch (Exception e) {
            e.printStackTrace();
            str = "upload fail";
        }
        return str;
    }
}
```

代码说明：

文件上传的表单提交需要遵循一定的 HTTP 请求头格式，代码中的"\r\n"、"--"、"*****"等字符串是为了配合这些格式而准备的。

在准备好上传方法后，需要在界面上调用这个方法，并用一个信息框显示服务器端传回来的信息，代码如下：

```
public class FileUpload extends Activity {
    private Button button;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.main);
        button = (Button) findViewById(R.id.button);   //实例化 Button 对象
        button.setOnClickListener(new OnClickListener() {
                                        //为 Button 对象添加监听
            public void onClick(View v) {
                showDialog(new ConnectWeb().uploadFile());
                                        //调用信息框，以上传方法的返回值为参数
            }
        });
    }

    private void showDialog(String mess) {             //显示上传信息框
        new AlertDialog.Builder(FileUpload.this).setTitle("Message")
                                                       //设置信息框标题
        .setMessage(mess).setNegativeButton("确定",
                                               //设置显示信息及添加确定按钮
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog,int which) {
                }
        }).show();
    }
}
```

程序运行效果如图 11.15 所示。



图 11.15　文件上传

第 12 章　Google 地图服务

作为 Google 公司开发的手机操作系统，Android 当然必不可少地支持 Google 的各项应用，其中应用最多的当属 Google Map 应用。本章中，就来研究一下 Android 应用程序中丰富多彩的地图应用。

12.1　获得 Android Maps API Key

要在 Android 应用程序中使用 Google Map 需要向 Google 申请一个 Android Maps API Key，这个 key 是和开发环境的计算机中 keystore 的 MD5 码绑定，因而是唯一的。一台机器申请的 key 可以在这台机器上开发的所有应用程序中使用。申请 Android Maps API Key 的步骤如下：

首先要找到开发环境中的 keystore 文件，在 MyEclipse 中，选择 Window|Preferences 命令，将会出现 Preferences 窗口，选择 Android|Build 命令，在 Default debug keystore 中可以看到 keystore 文件的位置，如图 12.1 所示。

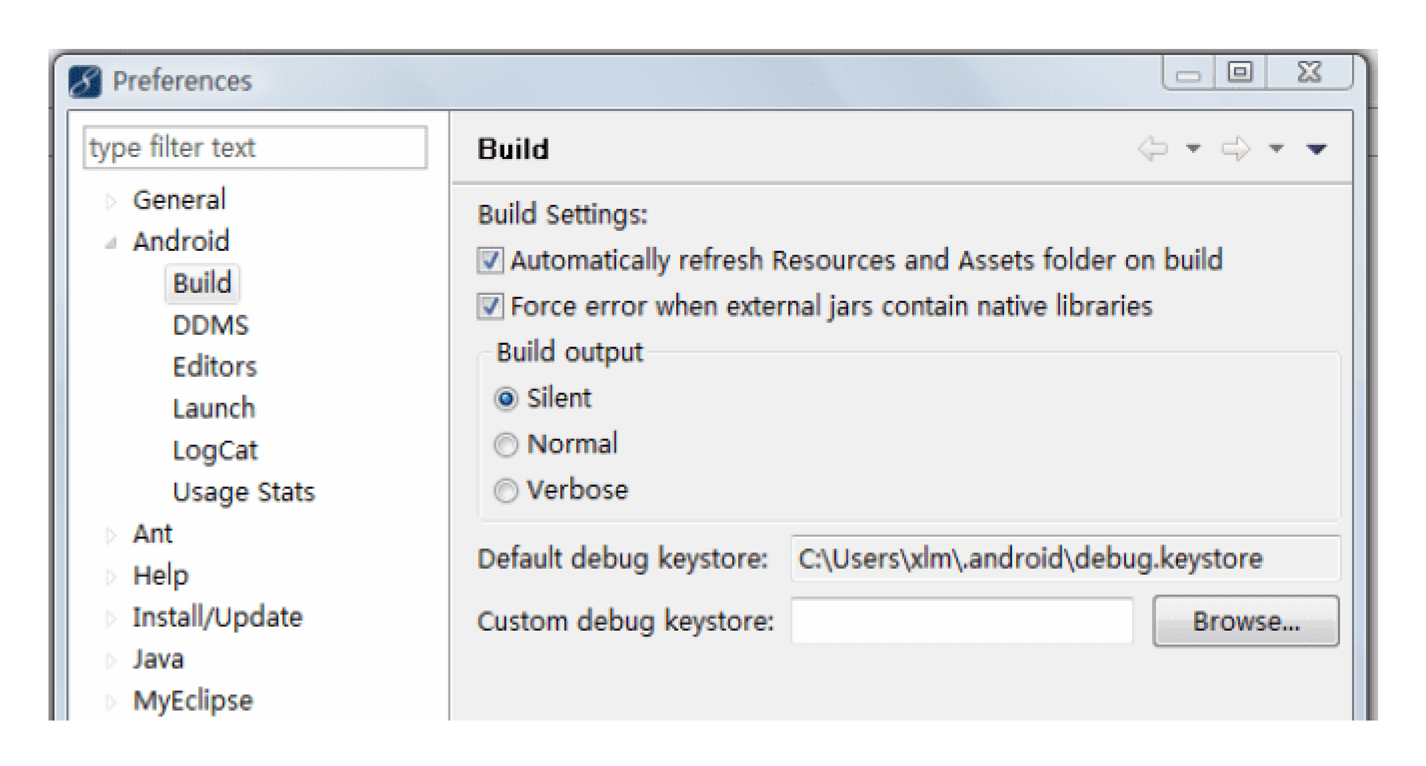


图 12.1　keystore 文件位置

然后，要生成 keystore 文件的 MD5 码。使用 cmd 命令进入 Dos 命令行状态，在本机的 JDK 的 bin 文件夹下输入命令：keytool-list-keystore “keystore 文件位置”，就可以生成 MD5 码，效果如图 12.2 所示。

最后，根据生成的 MD5 码，在 Google 网站上申请 Android Maps API Key，申请的步骤非常简单，网址是：http://code.google.com/intl/zh-CN/android/maps-api-signup.html，如图 12.3 所示。

为了显示地图，在创建模拟器的时候，需要选择 Google APIs，如图 12.4 所示。

在创建 Android 项目的时候，也需要选择 Google APIs，如图 12.5 所示。

```
C:\Program Files\Java\jdk1.6.0_17\bin>keytool -list -keystore C:\Users\xlm\.andr
oid\debug.keystore
输入keystore密码:

*****************  警告 警告 警告  *****************
*保存在您的 keystore 中数据的完整性  *
* 尚未被验证!  为了验证其完整性,  *
* 您必须提供您 keystore 的密码。                  *
*****************  警告 警告 警告  *****************

Keystore 类型: JKS
Keystore 提供者: SUN

您的 keystore 包含 1 输入

androiddebugkey, 2010-7-13, PrivateKeyEntry,
认证指纹 (MD5): CB:D2:99:F9:28:47:E9:E0:6E:F7:11:54:64:F6:08:90

C:\Program Files\Java\jdk1.6.0_17\bin>
```

图 12.2　生成 MD5 码

Google Maps API

Google Code Home > Google Maps API > Google Maps API Signup

Thank you for signing up for an Android Maps API key!

Your key is:

0u3n0rXoINNh5EdPDbnokQ82p09_vB7Gp_zR8Mw

This key is good for all apps signed with your certificate whose fingerprint is:

CB:D2:99:F9:28:47:E9:E0:6E:F7:11:54:64:F6:08:90

Here is an example xml layout to get you started on your way to mapping glory:

```
<com.google.android.maps.MapView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:apiKey="0u3n0rXoINNh5EdPDbnokQ82p09_vB7Gp_zR8Mw"
    />
```

Check out the API documentation for more information.

图 12.3　获得 Android Maps API Key

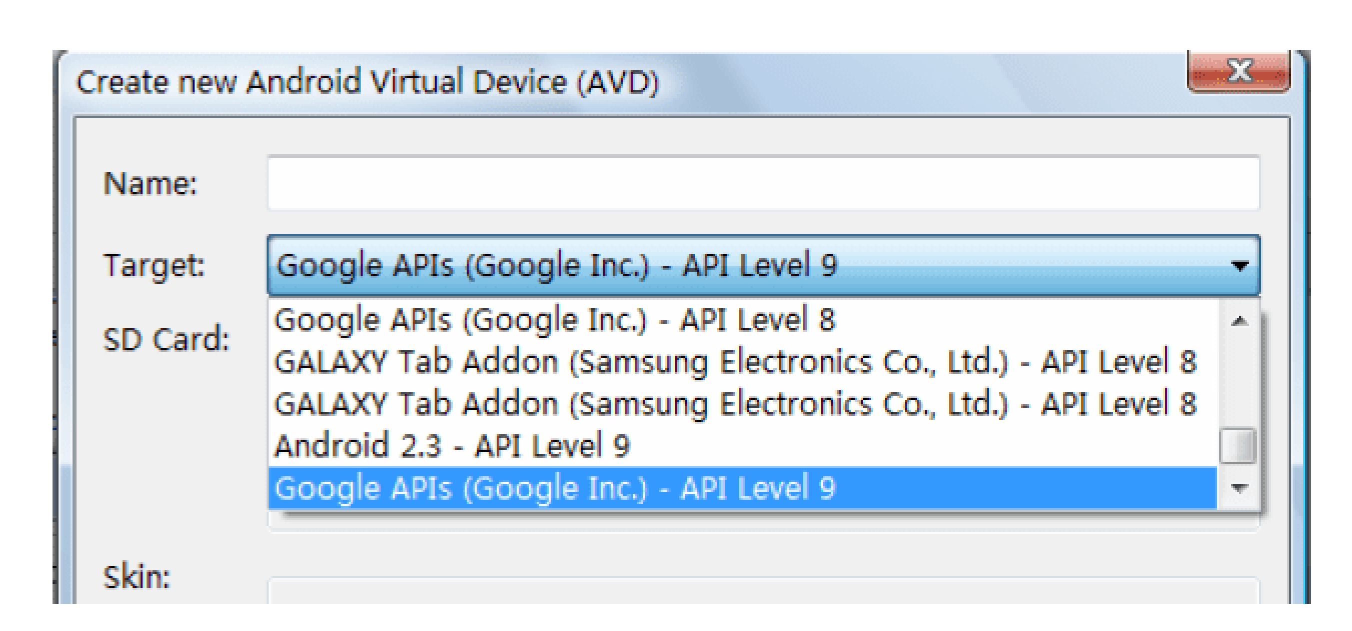


图 12.4　创建模拟器

Build Target

Target Name	Vendor	Platform	API ...
Android 1.5	Android Open Source Project	1.5	3
Google APIs	Google Inc.	1.5	3
Android 1.6	Android Open Source Project	1.6	4
Google APIs	Google Inc.	1.6	4
Android 2.0	Android Open Source Project	2.0	5
Google APIs	Google Inc.	2.0	5
Android 2.0.1	Android Open Source Project	2.0.1	6
Google APIs	Google Inc.	2.0.1	6
Android 2.1-up...	Android Open Source Project	2.1-up...	7
Google APIs	Google Inc.	2.1-up...	7
Android 2.2	Android Open Source Project	2.2	8
Google APIs	Google Inc.	2.2	8
GALAXY Tab A...	Samsung Electronics Co., Ltd.	2.2	8
GALAXY Tab A...	Samsung Electronics Co., Ltd.	2.2	8
Android 2.3	Android Open Source Project	2.3	9
☑ Google APIs	Google Inc.	2.3	9

图 12.5　创建地图 Android 项目

项目创建后，需要在 AndroidManifest.xml 文件中添加如下语句：

```
<uses-library android:name="com.google.android.maps" />
```

12.2　使用 MapView 显示地图

12.2.1　加载默认地图

在 Android 应用程序中显示地图的控件是 MapView，在 XML 文件中定义 MapView 控件的时候，就要用到上一节中申请的 key，定义代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android=http://schemas.android.com/apk/res/android
                                              <!--线形布局-->
  android:orientation="vertical"              <!--垂直摆放控件-->
  android:layout_width="wrap_content"         <!--自适应宽度-->
  android:layout_height="wrap_content">       <!--自适应高度-->
<view class="com.google.android.maps.MapView" <!--MapView 控件-->
    android:id="@+id/map"                     <!--控件 id-->
    android:layout_height="fill_parent"       <!--控件适应屏幕高度-->
    android:layout_width="fill_parent"        <!--控件适应屏幕宽度-->
    android:layout weight="1"                 <!--占据宽度比例-->
    android:enabled="true"                    <!--可操作为 true-->
    android:clickable="true"                  <!--可单击为 true-->
    android:apiKey="0u3n0rXoINNh5EdPDbmokQ82p09_vB7Gp_zR8Mw"/>
                                          <!--设置 Android Maps API Key-->
</LinearLayout>
```

然后在 Java 类中实例化 MapView 控件，代码如下：

```
public class MyWebView extends MapActivity{
    private MapView map;                             //定义 MapView 控件
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);                //加载 XML 文件
       map=(MapView)findViewById(R.id.map);          //实例化 MapView 控件
    }
    protected boolean isRouteDisplayed() {           //重写方法
        return false;
    }
}
```

代码说明：

- 要在应用程序中使用 MapView 控件，需要继承 com.google.android.maps.MapActivity 类，该类是 android.app.Activity 类的子类。
- 在 AndroidManifest.xml 文件中添加<uses-library android:name="com.google.android.maps" />和<uses-permission android:name="android.permission.INTERNET" />。

程序运行效果如图 12.6 所示。



图 12.6　MapView 运行效果

12.2.2　加载自定义地图

上面的例子非常简单，只是实例化了 MapView 控件，通过 MapView 显示出了地图。地图显示的位置是美国，这是 MapView 控件的默认设置。而且目前我们可以执行拖曳地图的操作，但是无法缩放地图。在实际的应用程序中，需要按需求改变地图初始的显示位置，比如初始显示北京，并且可以让用户拖曳缩放地图，这需要对程序进行修改补充，代码如下：

```
public class MyWebView extends MapActivity{
    private MapView map;                              //定义 MapView 控件
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);                 //加载 XML 文件
       map=(MapView)findViewById(R.id.map);           //实例化 MapView 控件
       map.setBuiltInZoomControls(true);              //设置内置缩放工具可用
       GeoPoint        center=new        GeoPoint((int)(39.9067452*1E6),
(int)(116.391177*1E6)); //实例化 GeoPoint 对象
       MapController mcontrol=map.getController();//获取 MapController 对象
       mcontrol.setCenter(center);                    //设置地图中心位置
       mcontrol.setZoom(10);                          //设置地图缩放级别
    }
    protected boolean isRouteDisplayed() {            //重写父类方法
        return false;
    }
}
```

代码说明：

- GeoPoint 是用来存储经纬度的类。
- 普通的地理数据的经纬度是小数的形式，需要分别乘以 1E6 才能生成一个 GeoPoint 对象。

❑ MapController 提供了一系列来移动和缩放地图的类，在后面的章节中将会详细介绍。

运行上面的代码，会发现地图已经以北京为中心显示了。单击地图后，会出现一个放大和缩小地图的工具条，程序运行效果如图 12.7 所示。

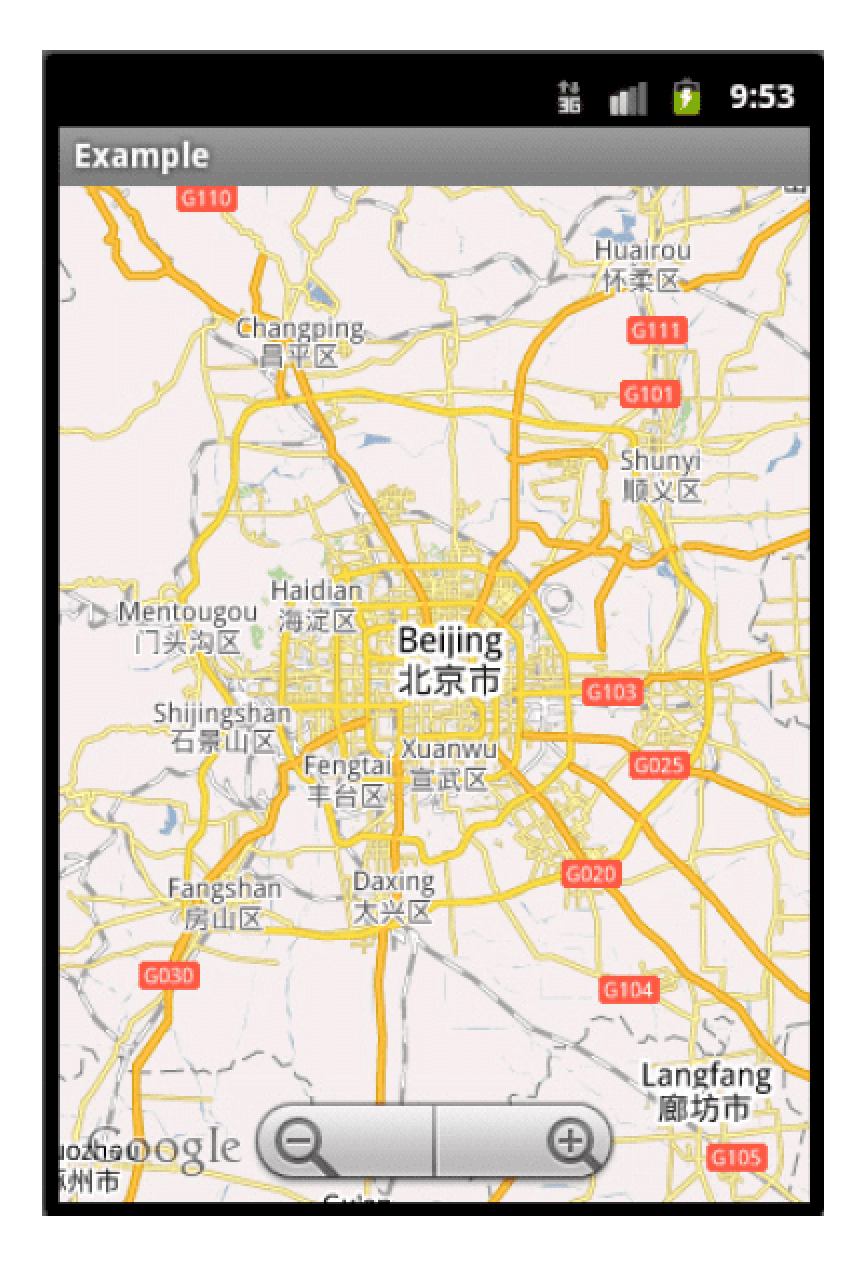


图 12.7　改变地图中心位置

12.2　在地图上做标记

和网站上的 Google Map 一样，手机端的地图应用也允许用户在地图上添加自己的图标。上一节中介绍了如何展示地图，本节的目标是研究一下如何在 Android 应用程序中添加自定义的地图图标。要将图标显示在地图上，需要 com.google.android.maps.Overlay 类的帮助。

要了解的是，网站或是手机端所看到的电子地图，其实是由多个层叠加在一起形成的。其中有负责绘制道路的“层”，有负责绘制文字标注的“层”等。

Overlay 是一个可以覆盖在地图现有层之上的一个层。开发者可以使用 Overlay 类提供的方法，自定义一个层，与其他的层结合在一起，达到在地图上添加标记的目的。使用 Overlay 类添加标注的基本流程是：先将一个给定的地理坐标转换成屏幕上的 X/Y 坐标，然后将需要的内容绘制在 X/Y 坐标位置上。

下面的例子是将一个小图片绘制在给定的坐标上，代码如下：

```
public class Marker extends Overlay{
    private GeoPoint point;
    private Bitmap bmp;
    public Marker(GeoPoint point, Bitmap bmp){
                                //构造函数，参数为绘制标注的坐标点和标注图片
        this.point = point;
        this.bmp = bmp;
    }

    public void draw(Canvas canvas, MapView mapView, boolean shadow) {
                                //重写 draw()方法，绘制图片
```

```
        Projection projection = mapView.getProjection();
                                                    //获取 Projection 类对象
        Point pos = projection.toPixels(point, null);//转化地理坐标到 X/Y 坐标
        canvas.drawBitmap(bmp, pos.x, pos.y, null); //绘制图片
    }
}
```

代码说明：

- draw()方法是 Overlay 类用来绘制内容的方法，会自动调用。
- Projection 是一个工具接口，负责地理坐标系统和屏幕 X/Y 坐标系统之间的转化，可以通过 MapView 的 getProjection()方法获取实例。
- com.google.android.maps.Projection 的 toPixels()方法将一个地理坐标转化为 X/Y 坐标，该方法有两个参数，第 1 个为 com.google.android.maps.GeoPoint 类型，是需要转的地理坐标；第 2 个参数为 android.graphics.Point 类型，保存转换后的 X/Y 坐标，如果为 null，则新建一个 android.graphics.Point 类对象。
- android.graphics.Point 类用来保存 X/Y 坐标信息。
- android.graphics.Canvas 负责绘制内容，它提供了一组方法可以绘制文字、图形、图片等内容，在后续的章节将会有更详细的案例介绍。
- drawBitmap()是 Canvas 绘制图片的方法，有多种重载形式。这里采用的形式带有 4 个参数，第 1 个是 android.graphics.Bitmap 类型，表示要绘制的图片；第 2 个是 float 类型，表示图片的 left 值；第 3 个是 flaot 类型，表示图片的 top 值；第 4 个是 android.graphics.Paint 类型，表示描绘时的画笔设置，该参数的详细设置在后续的章节会有详细的介绍，如果为 null，表示使用默认设置。

接下来，将上一节展示地图的程序修改一下，加上准备好的地图标注，代码如下：

```
public class MyWebView extends MapActivity{
    private MapView map;                        //定义 MapView 控件
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);           //加载 XML 文件
       map=(MapView)findViewById(R.id.map);     //实例化 MapView 控件
       map.setBuiltInZoomControls(true);        //设置内置缩放工具可用
       GeoPoint center=new GeoPoint((int)(39.9067452*1E6), (int)
       (116.391177*1E6));                       //实例化 GeoPoint 对象
       MapController mcontrol=map.getController();//获取 MapController 对象
       mcontrol.setCenter(center);              //设置地图中心位置
       mcontrol.setZoom(10);                    //设置地图缩放级别
       GeoPoint point=new GeoPoint((int)(39.9067452*1E6), (int)
       (116.391177*1E6));                       //设置标注坐标点
       Bitmap bmp = BitmapFactory.decodeResource(getResources(),
       R.drawable.flaggreen);                   //获取标注图片
       Marker marker = new Marker(point,bmp);   //实例化 Marker 对象
       map.getOverlays().add(marker);           //添加层
    }
    protected boolean isRouteDisplayed() {
        return false;
    }
}
```

程序运行效果如图 12.8 所示。

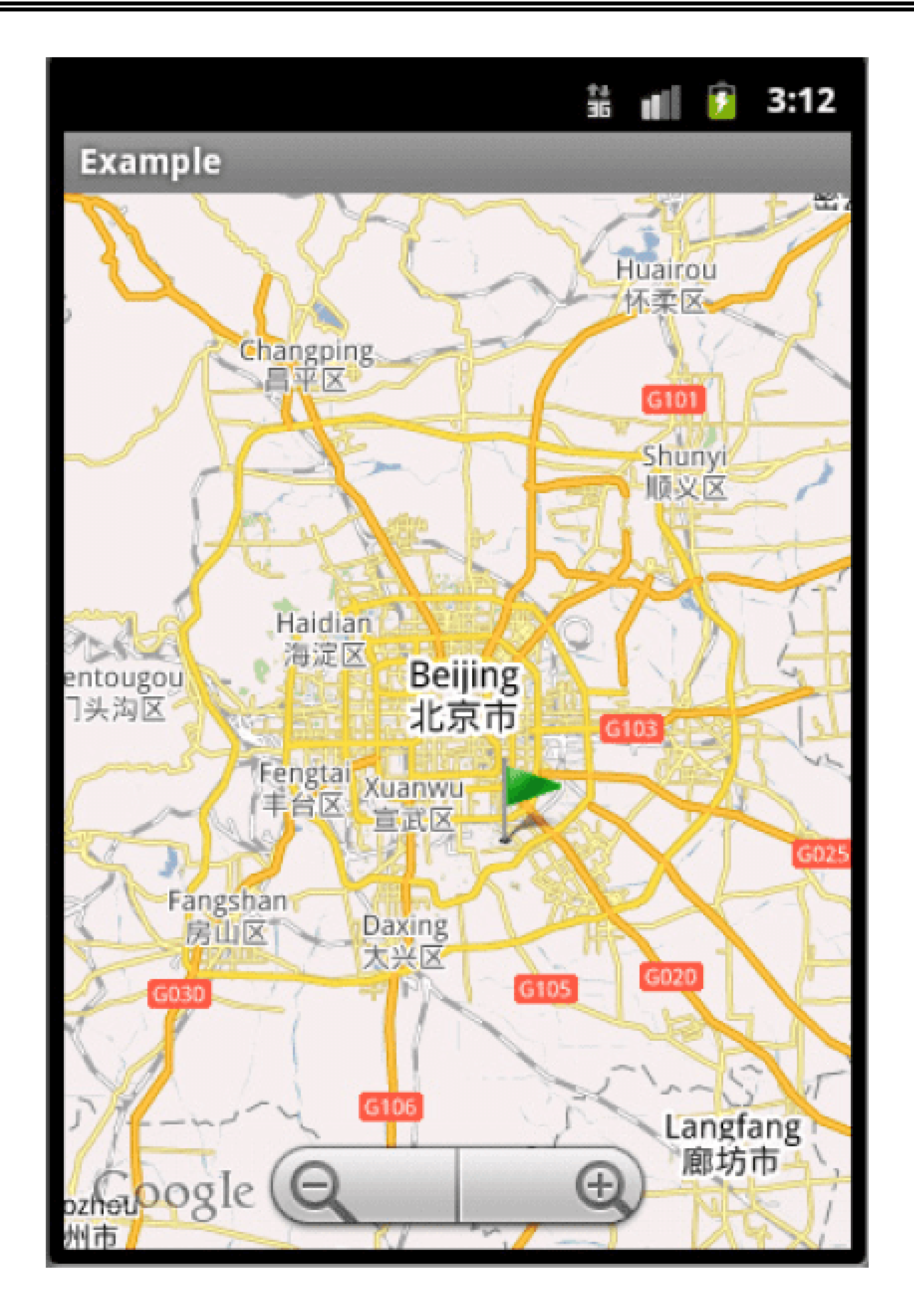


图 12.8　添加地图标注

12.3　地图标注响应单击事件

上一节中的例子简单实现了使用图符在地图上标注的功能。但是使用 Overlay 的对象来做地图标注存在着一些技术上的缺陷，最大的不便来自于开发者无法简便地获知使用者是否单击了标注，或是单击了哪个标注。因而 Overlay 大多是用于描绘地图的线段或是一些不需要处理的地图标注，Google APIs 提供了另外一个类 com.google.android.maps.ItemizedOverlay，用来处理那些和用户交互的标注。

ItemizedOverlay 是 Overlay 的子类，它创建并维护一个 OverlayItem 对象数组，Overlay 会为每个点绘制标注图像，并且负责把一个屏幕单击匹配到某个 OverlayItem 上去，分发焦点改变事件给备选的监听器。使用 ItemizedOverlay 的基本流程是：每一个地图标注都作为一个 OverlayItem 对象加入这个数组，然后再添加进 MapView 中去，ItemizedOverlay 提供了一套方法来识别和判断是哪一个标注被单击。

下面是准备 ItemizedOverlay 的例子，代码如下：

```
public class Marker extends ItemizedOverlay<OverlayItem> {
    //保存地图标注的 OverlayItem 集合
    private ArrayList<OverlayItem> markerList = new ArrayList<OverlayItem>();
    private Context context;
    public Marker(Drawable defaultMarker) {     //构造方法，定义标注的图片
        super(boundCenterBottom(defaultMarker));
        this.context=context;                   //定义 Toast 所属的 Context
    }
    protected OverlayItem createItem(int arg0){//创建并返回一个对象
        return markerList.get(arg0);
    }
    public int size() {                         //集合的大小
        return markerList.size();
```

```
    }
    public void addOverlay(OverlayItem overlay) {
        markerList.add(overlay);                    //将新的标注加入集合
        populate();                                 //在地图上绘制标注
    }
    protected boolean onTap(int i) {
        //显示被单击的标注的信息
        Toast.makeText(MyWebView.this, createItem(i).getTitle(),Toast.
        LENGTH LONG).show();
        return true;
    }
}
```

代码说明：

- OverlayItem 是 ItemizedOverlay 的基本组成部分，实例化一个 OverlayItem 对象需要一个坐标，和两个字符串做参数，具体的代码很快就会看到。
- ItemizedOverlay()的构造方法需要一个 android.graphics.drawable.Drawable 类型的参数，代表了绘制在地图上的标注的图像。所有加入 ItemizedOverlay 的 OverlayItem 对象都使用这个图像。
- addOverlay()是我们定义的方法，用于将一个 OverlayItem 对象加入集合。
- addOverlay 中调用的 populate()方法是 ItemizedOverlay 类的工具方法，当一个新的 OverlayItem 加入到集合中后，一定要调用该方法。Populate()方法会调用 createItem()方法在地图上描绘出标注。
- createItem()、size()、onTap()是重写父类的方法。
- createItem()方法会在新的 OverlayItem 被添加的时候在地图上创建一个新的标注，在本例中使用该方法返回集合中指定下标的 OverlayItem 对象。
- onTap()方法响应用户对标注单击操作，有两种重载方式。子类需重写带有一个整数类型参数的方法，当用户单击了地图标注后会触发此方法，整数参数就是被单击的标注在集合中的下标。
- ItemizedOverlay 是地图上的一个附加层，也是地图的一部分。当 onTap()方法响应了单击事件后，单击事件会继续向 ItemizedOverlay 层下地图传递。如果希望阻止这种事件传递，onTap()方法的返回值要返回 true。

接下来，就是准备在 MapView 上添加 ItemizedOverlay，代码如下：

```
public class MyWebView extends MapActivity {
    private MapView map;                                //定义 MapView 控件
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);                  //加载 XML 文件
        map = (MapView) findViewById(R.id.map);         //实例化 MapView 控件
        map.setBuiltInZoomControls(true);               //设置内置缩放工具可用
        //实例化 GeoPoint 对象
        GeoPoint center=new GeoPoint((int) (23.3497311 * 1E6),(int)
        (108.4699989 * 1E6));
        MapController mcontrol = map.getController();
                                                    //获取 MapController 对象
        mcontrol.setCenter(center);                 //设置地图中心位置
        mcontrol.setZoom(10);                       //设置地图缩放级别
        Drawable drawable = this.getResources().getDrawable
        (R.drawable.flaggreen);                     //实例化 Drawable 对象
```

```
        Marker marker = new Marker(drawable, MyWebView.this);
                                            //实例化 Marker
        GeoPoint point1 = new GeoPoint((int) (23.3497311 * 1E6),(int)
        (108.4699989 * 1E6));
        OverlayItem overlayitem1 = new OverlayItem(point1, "图标一号", "");
                                        //实例化第一个 OverlayItem 对象
        GeoPoint point2 = new GeoPoint((int) (23.4727264 * 1E6),(int)
        (108.5281492 * 1E6));
        OverlayItem overlayitem2 = new OverlayItem(point2, "图标二号", "");
                                        //实例化第二个 OverlayItem 对象
        marker.addOverlay(overlayitem1);
                                //将 OverlayItem 对象添加进 ItemizedOverlay
        marker.addOverlay(overlayitem2);
                                //将 OverlayItem 对象添加进 ItemizedOverlay
        map.getOverlays().add(marker); //将 ItemizedOverlay 加入 MapView
    }
    protected boolean isRouteDisplayed() {
        return false;
    }
}
```

代码说明：

- 实例化一个 OverlayItem 的时候，可以将一些标示或描述这个 OverlayItem 对象的信息写进它的 title 和 snippet 属性中去。在 ItemizedOverlay 类的 onTap()方法中，取出了被单击的 OverlayItem 的 title 值显示。
- 所有的 OverlayItem 对象都通过 addOverlay()方法加入到 ItemizedOverlay 类中的集合里去，最后由 MapView 来添加 ItemizedOverlay 类对象。

程序运行的效果如图 12.9 所示。



图 12.9　ItemizedOverlay 运行效果

12.4　自定义地图提示信息

前而各节中的例子使用了 Toast 来显示被单击标注的信息，这种方式在标注比较多的时候，不能让用户直观地看到究竟是哪一个标注被单击，如果提示信息能够出现在标注的旁边，效果就会好很多。本节的目标，就是自定义信息框并让信息框显示在标注的旁边，效果如图 12.10 所示。



图 12.10　自定义地图提示信息

在 Google APIs 中，并没有提供类似 Web 网站上那样的信息提示框，因此需要通过其他手段来实现这一效果。

上面图片中的提示框使用的是 Overlay 类，通过 Canvas 对象描绘了一个有透明效果的圆角矩形框，并在这个矩形框内写上文字。这个矩形框的位置是和被单击的标注的坐标相关联的，具体代码如下：

```
public class MyWebView extends MapActivity {
    private MapView map;                          //定义 MapView 控件
    private TextOverlay to;                       //定义信息框对象
    private int xy[]=new int[4];                  //定义信息框四角像素位置数组
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);            //加载 XML 文件
        map = (MapView) findViewById(R.id.map);//实例化 MapView 控件
        map.setBuiltInZoomControls(true);         //设置内置缩放工具可用
        //实例化 GeoPoint 对象
        GeoPoint center=new GeoPoint((int) (23.3497311 * 1E6),(int)
        (108.4699989 * 1E6));
        MapController mcontrol = map.getController();//获取 MapController 对象
```

```
        mcontrol.setCenter(center);                 //设置地图中心位置
        mcontrol.setZoom(10);                       //设置地图缩放级别
        Drawable drawable = this.getResources().getDrawable
        (R.drawable.flaggreen);                     //实例化 Drawable 对象
        Marker marker = new Marker(drawable, MyWebView.this);//实例化 Marker
        GeoPoint point1 = new GeoPoint((int) (23.3497311 * 1E6),(int)
        (108.4699989 * 1E6));
        OverlayItem overlayitem1 = new OverlayItem(point1, "图标一号", "");
                                            //实例化第一个 OverlayItem 对象
        GeoPoint point2 = new GeoPoint((int) (23.4727264 * 1E6),(int)
        (108.5281492 * 1E6));
        OverlayItem overlayitem2 = new OverlayItem(point2, "图标二号", "");
                                            //实例化第二个 OverlayItem 对象
        marker.addOverlay(overlayitem1);
                                //将 OverlayItem 对象添加进 ItemizedOverlay
        marker.addOverlay(overlayitem2);
                                //将 OverlayItem 对象添加进 ItemizedOverlay
        map.getOverlays().add(marker); //将 ItemizedOverlay 加入 MapView
    }

    protected boolean isRouteDisplayed() {
        return false;
    }

    public void setXY(int x1,int y1,int x2,int y2){
                                        //保存当前信息框四角像素坐标
        xy[0]=x1;
        xy[1]=x2;
        xy[2]=y1;
        xy[3]=y2;
    }

    public boolean dispatchTouchEvent(MotionEvent ev) {
        int cx=(int)ev.getX();              //获取单击屏幕的 x 坐标
        int cy=(int)ev.getY();              //获取单击屏幕的 y 坐标
        if((cx<xy[0] || cx>xy[1]) || (cy<xy[2] || cy>xy[3])){
                                            //判断单击屏幕位置的坐标是否在信息框中
            if(to!=null){                       //判断信息框对象是否为空
                map.getOverlays().remove(to);   //从屏幕移除信息框
                to=null;                        //置空信息框对象
            }
        }
        return super.dispatchTouchEvent(ev);
    }

    public class Marker extends ItemizedOverlay<OverlayItem> {//标注内部类
        //保存地图标注的 OverlayItem 集合
        private ArrayList<OverlayItem> markerList = new ArrayList
        <OverlayItem>();
        public Marker(Drawable defaultMarker) {     //构造方法，定义标注的图片
            super(boundCenterBottom(defaultMarker));
        }

        protected OverlayItem createItem(int arg0) {   //创建并返回一个对象
            return markerList.get(arg0);
        }
```

```
        public int size() {                              //集合的大小
            return markerList.size();
        }

        public void addOverlay(OverlayItem overlay) {
            markerList.add(overlay);                     //将新的标注加入集合
            populate();                                  //在地图上绘制标注
        }

        protected boolean onTap(int i) {
            to=new TextOverlay(createItem(i).getTitle(),createItem(i).
            getPoint());                                 //实例化信息框对象
            map.getOverlays().add(to);                   //添加信息框到地图
            return true;
        }
    }

    public class TextOverlay extends Overlay{            //信息框内部类
        private String str;                              //信息字符串
        private GeoPoint p;                        //需弹出信息框的标注坐标位置
        public TextOverlay(String str,GeoPoint p){
            this.str=str;
            this.p=p;
        }

        public boolean draw(Canvas arg0, MapView arg1, boolean arg2, long arg3){
            if(!arg2){                             //判断是否描绘阴影层
                Paint paint=new Paint();          //定义用户描绘文字的 Paint 对象
                paint.setTextSize(14);             //设置字号
                paint.setColor(Color.WHITE);       //设置字体颜色
                paint.setAntiAlias(true);          //消除信息框锯齿
                paint.setFakeBoldText(true);       //消除文字锯齿
                Paint bpaint=new Paint();          //定义用户描绘边框的 Paint 对象
                bpaint.setColor(Color.BLACK);      //设置背景色
                bpaint.setAntiAlias(true);         //消除文字锯齿
                bpaint.setAlpha(150);              //设置透明度
                Point temp=map.getProjection().toPixels(p, null);
                                                   //将标注坐标转化为 x/y 坐标
                int x1=temp.x+15;                  //计算信息框左上角 x 值
                int y1=temp.y-30;                  //计算信息框左上角 y 值
                int count=str.length();
                int x2=x1+(count+2)*14;            //计算信息框右下角 x 值
                RectF boval=new RectF(x1,y1,x2,y1+30);//计算信息框右下角 y 值
                setXY(x1,y1+50,x2,y1+80);
                arg0.drawRoundRect(boval, 10, 10, bpaint);//描绘信息框
                arg0.drawText(str, x1+14, y1+20, paint);  //描绘信息框文字
            }
            return super.draw(arg0, arg1, arg2, arg3);
        }
    }
}
```

代码说明：

- ❑ 示例中代码包含 Marker 和 TextOverlay 两个内部类。Marker 类负责创建地图标注，TextOverlay 负责创建信息提示框。
- ❑ 信息提示框的构造方法有两个参数，其一是被单击的标注点坐标，信息提示框出

现的位置是和被单击的标注坐标绑定的，所以在创建信息框对象的时候，要将该标注的坐标作为参数传递进来。另一个参数是字符串，即需要在信息框上显示的文字信息。

- 信息框由两部分组成，一部分是半透明的圆角矩形框，一部分是信息文字。圆角矩形框由 android.graphics.Canvas 类的 drawRoundRect()方法绘制，文字由 android.graphics.Canvas 类的 drawText()方法绘制。
- drawRoundRect()方法可以绘制一个圆角矩形，有 4 个参数。第 1 个参数是 android.graphics.RectF 类型，负责定义矩形的位置及尺寸；第 2 个参数是 float 类型，用于定义矩形圆角的 x 半径；第 3 个参数是 float 类型，用于定义矩形圆角的 y 半径；第 4 个参数是 android.graphics.Paint 类型，用于定义描绘的参数。
- 实例化一个 android.graphics.RectF 类对象需要 4 个参数，根据 Android API 的说法是矩形居左、上、右、下四个边的坐标。其实可以简单地理解为矩形左上角和右下角的 x/y 坐标。
- 代码中将被单击的标注地理坐标转化为 x/y 坐标后，需要根据这个坐标计算出信息框的左上角。整个屏幕的 x/y 坐标系的原点即（0，0）点是屏幕左上角，而程序需要将信息框显示在标注点的左侧与标注点等高的位置，所以将标注点的 x 坐标增大向右平移，y 坐标减少向上平移，生成的新坐标点就是信息框的左上角坐标；根据设定的文字的字号计算出单个文字大约有 14 个像素的宽度，然后再根据需显示信息的文字个数加上左右两端的留白，计算出整个信息框的宽度，即信息框右下角的 x 坐标，整个信息框的高度就是计算左上角 y 坐标时的平移度，左上角的 y 坐标减去这个平移度就是右下角的 y 坐标。
- android.graphics.Paint 类负责设定绘制的文字、图形、图像的样式和颜色的参数信息。
- drawText()方法有多种重载形式，本例中采用的是带 4 个参数的方式。第 1 个参数是字符串类型，定义应需描绘的文字；第 2 个参数是 float 类型，定义开始描绘文字的 x 坐标；第 3 个参数是 float 类型，定义开始描绘文字的 y 坐标；第 4 个参数是 android.graphics.Paint 类型，用于定义描绘的参数。
- 当单击其他标注的时候，原有的信息框要消失。所以将信息框的左上角和右下角的坐标值保存在一个数组内。当用户单击屏幕的时候，判断单击位置是否在信息框范围内，如果不在就从地图移除信息框。这个操作写在 MapActivity 的 dispatchTouchEvent 方法内。

12.5 在地图上显示当前位置

12.5.1 获取真机 GPS 信号

在地图上显示用户当前位置，是手机地图应用中很有意思的一部分。基本的思路是利用手机的 GPS 或信号基站来确定手机当前的坐标位置，然后以标注的形式在地图上表现出来。

要特别注意的是，需要在应用程序中添加相应的权限，来调用手机的 GPS 定位功能，代码如下：

```
<uses-permission android:name="android.permission.INTERNET" />
```

```
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

定位功能实现的基本流程是：实例化一个 LocationManager，注册一个 LocationListener 监听位置变化，获取改变后的位置坐标。Android SDK 提供了一组在 android.location 包下的 API 支持上述操作，代码如下：

```
public class MyWebView extends MapActivity {
    private MapView map;                          //定义 MapView 对象
    private LocationManager locationManager;      //定义 LocationManager 对象
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);            //加载界面 XML 文件
        map = (MapView) findViewById(R.id.map);//实例化 MapView 对象
        map.setBuiltInZoomControls(true);         //设置内置缩放工具可用
        GeoPoint center = new GeoPoint((int) (39.9067452 * 1E6),
                (int) (116.391177 * 1E6));        //定义地图中心点坐标
        MapController mcontrol = map.getController();
                                                  //获取 MapController 对象
        mcontrol.setCenter(center);               //设置地图中心点
        mcontrol.setZoom(10);                     //设置地图缩放级别
        locationManager = (LocationManager) this
                .getSystemService(Context.LOCATION_SERVICE);
                                              //实例化 LocationManager 对象
        locationManager.requestLocationUpdates(LocationManager.
                GPS_PROVIDER,1000, 0.0001f, locationListener);
                                              //注册 LocationListener 监听器
    }

    public void getMyLoc() {
        Criteria criteria = new Criteria();      //实例化 Criteria 对象
        criteria.setAccuracy(Criteria.ACCURACY_FINE);
        criteria.setAltitudeRequired(false);
        criteria.setBearingRequired(false);
        criteria.setCostAllowed(false);
        criteria.setPowerRequirement(Criteria.POWER_LOW);

        Location location = locationManager      //获取当前位置
                .getLastKnownLocation(locationManager.getBestProvider
                (criteria,true));
        if (location == null) {
            Toast.makeText(MyWebView.this, "没有 GPS 信号", Toast.LENGTH_
            SHORT).show();
        }
        else {
            GeoPoint point = new GeoPoint((int) (location.getLatitude() * 1E6),
                    (int) (location.getLongitude()*1E6));//获取当前位置坐标
            Bitmap bmp = BitmapFactory.decodeResource(getResources(),
                    R.drawable.flagred);                 //准备标注图片
            Marker marker = new Marker(point, bmp);      //实例化 Marker 对象
            map.getOverlays().add(marker);               //添加地图层
            map.invalidate();                            //刷新地图
        }
    }

   private final LocationListener locationListener = new LocationListener()
```

```
{   // LocationListener 监听器
        public void onLocationChanged(Location location) {
                                                    //当位置坐标变化时触发
            getMyLoc();                             //获取当前位置
        }
        public void onProviderDisabled(String provider) {
                                                    //当用户禁用功能时调用
        }

        public void onProviderEnabled(String provider) {
                                                    //当用户启用功能时调用
        }

        public void onStatusChanged(String provider, int status, Bundle
        extras) {                                   //当功能状态改变时调用
        }
    };

    protected boolean isRouteDisplayed() {
        return false;
    }

    public class Marker extends Overlay {
        private GeoPoint point;
        private Bitmap bmp;
        public Marker(GeoPoint point, Bitmap bmp) {
                                    //构造函数，参数为绘制标注的坐标点和标注图片
            this.point = point;
            this.bmp = bmp;
        }

        public void draw(Canvas canvas, MapView mapView, boolean shadow)
{                                               //重写 draw()方法，绘制图片
            Projection projection = mapView.getProjection();
                                                //获取 Projection 类对象
            Point pos = projection.toPixels(point, null);
                                                //转化地理坐标到 X/Y 坐标
            canvas.drawBitmap(bmp, pos.x, pos.y, null);     //绘制图片
        }
    }
}
```

代码说明：

- android.location.LocationManager 类似访问系统定位服务的管理者，开发者不能直接实例化这个类的对象，而需要使用 Context.getSystemService(Context.LOCATION_SERVICE)方法获取它的实例。
- requestLocationUpdates()方法用来注册实时监听手机位置，有多种重载形式。本例中使用的是带 4 个参数的方式。第 1 个参数是 String 类型，定义的是定位技术的种类；第 2 个参数是 long 类型，定义的是两次监听操作的时间间隔；第 3 个参数是 float 类型，定义的是触发监听器方法的最小距离；第 4 个参数是 LocationListener 类型，定义的是监听位置变化的监听器。
- android.location.LocationListener 负责监听位置的变化，它由 4 个方法需要被重写，最重要的是 onLocationChanged()方法。LocationListener 会每隔一段时间检查一下当前的位置，时间间隔的单位是毫秒，在 LocationManager 的 requestLocationUpdates()

方法的第 2 个参数设定数值。当位置变化超过一定范围后，触发 onLocationChanged()方法。位置变化范围的数值在 LocationManager 的 requestLocationUpdates()方法的第 3 个参数设定。

- LocationManager 的 getLastKnownLocation()方法可以获取当前手机最后一个已知位置。如果当前定位方式不可用，如无法获得 GPS 信号，该方法返回一个 null；否则，返回一个 Location 对象。
- android.location.Location 类对象代表一个地理位置，包含经纬度、方向、高度等信息。本例中使用 getLatitude()方法和 getLongitude()方法获取经纬度的数值。

程序运行效果如图 12.11 所示。

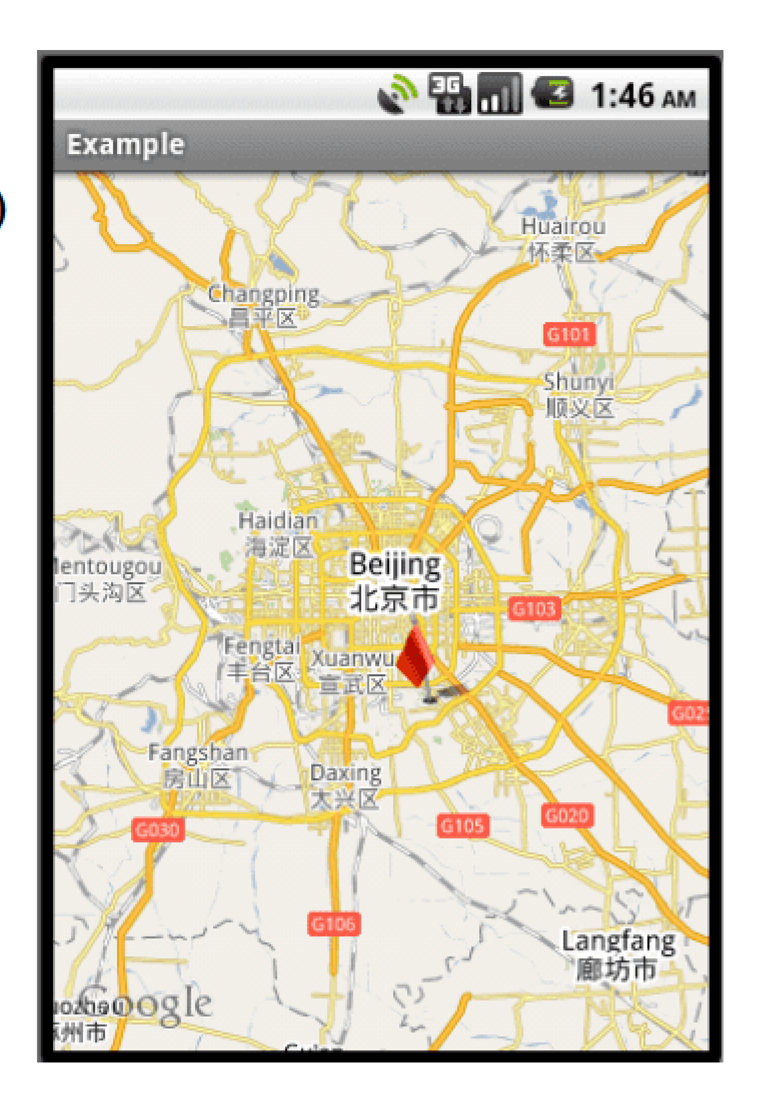


图 12.11　显示当前位置

12.5.2　模拟器获取地理坐标

如果希望在模拟器上调试定位程序，我们可以使用模拟器上提供的方法输入一个模拟坐标。在 MyEclipse 的 DDMS 视图下，选择 Emulator Controls，效果如图 12.12 所示。

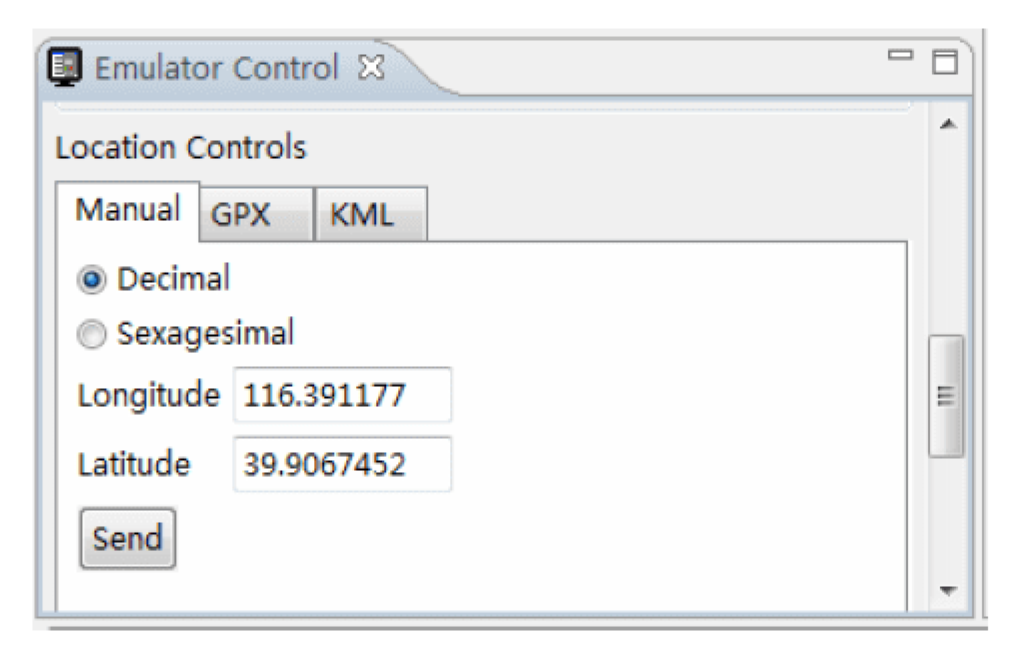


图 12.12　模拟器输入坐标

需要说明的是，定位程序在手机上运行的时候，会出现“定位不准”的现象，即图符显示的位置与实际位置有一定的偏差。这并不是我们获取的坐标信息有误，而是因为根据我国法律，电子地图在展现出来的时候都要做偏移处理，而我们直接从 GPS 等信号源获取的定位坐标信息是没有经过偏移处理的，这之间的差别造成了我们视觉上的“定位不准”。

12.6　地理查询与逆地理查询

在地图应用中，经常有些需求：根据一个地址名称查询该地址的地理坐标，即地理查询；或者反过来，根据一个地理坐标查询该坐标地址，即逆地理查询。

在 Android SDK 中，上述操作被封装在 android.location.Geocoder 类中。需要说明的是，具体的信息查询的操作是封装在后台服务中发送到 Google Map Service 去查询的，Geocoder 类提供了方法让开发人员可以获得查询的结果。

12.6.1　地理查询

地理查询的代码如下：

```
public class MyWebView extends MapActivity {
    private MapView map;                              //定义 MapView 控件
    private GeoPoint center;                          //定义地图中心点
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);                //加载 XML 文件
        map = (MapView) findViewById(R.id.map);       //实例化 MapView 控件
        map.setBuiltInZoomControls(true);             //设置内置缩放工具可用
        //实例化地图中心 GeoPoint 对象
        center = new GeoPoint((int) (39.9067452 * 1E6), (int) (116.391177 * 1E6));
        final MapController mcontrol = map.getController();
                                                    //获取 MapController 对象
        mcontrol.setCenter(center);                 //设置地图中心位置
        mcontrol.setZoom(10);                       //设置地图缩放级别

        getAddressByLocation();
    }

    protected boolean isRouteDisplayed() {
        return false;
    }

    public void getAddressByLocation(){
        Geocoder gc = new Geocoder(this, Locale.CHINA);//实例化 Geocoder 对象
        try{
            //根据坐标获取地址 Address 对象集合
            List<Address> address =gc.getFromLocation(39.9067452,116.
            391177, 1);
            StringBuilder sb = new StringBuilder();
            if (address.size() > 0){
             Address adsLocation = address.get(0);//获取集合中第一个地址对象
             sb.append(adsLocation.getAdminArea()).append("\n");
                                                  //获取行政区域名称
             sb.append(adsLocation.getCountryCode()).append("\n");
                                                  //获取国家代码
             sb.append(adsLocation.getCountryName()).append("\n");
                                                  //获取国家名称
             sb.append(adsLocation.getFeatureName()).append("\n");
                                                  //获取特征名称
             sb.append(adsLocation.getLocality()).append("\n");
                                                  //获取地方名称
             sb.append(adsLocation.getPhone()).append("\n");   //获取电话
             sb.append(adsLocation.getPostalCode()).append("\n");
                                                  //获取邮政编码
             sb.append(adsLocation.getPremises()).append("\n");
                                                  //获取房地名称
             sb.append(adsLocation.getThoroughfare());      //获取道路名称
           }
            Toast.makeText(MyWebView.this, sb.toString(),Toast.LENGTH_LONG).
            show();
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

代码说明：

- 实例化 android.location.Geocoder 对象需要两个参数，第 1 个为 android.content.Context 对象；第 2 个为 java.util.Locale 对象。
- Locale 类是用来描述不同国家语言组合的类，Locale.CHINA 常量返回的是中文语言环境。开发者也可以使用 Locale 类的 getDefault()方法获取默认的区域环境。如果使用默认环境，实例化 Geocoder 对象就只需要一个参数，即 Context 对象参数。
- Geocoder 类的 getFromLocation()方法会根据经纬度坐标返回一个结果集合。该方法需要 3 个参数，第 1 个是 double 类型，定义纬度值；第 2 个是 double 类型，定义经度值；第 3 个是 int 类型，定义返回结果的个数。
- android.location.Address 类是用来描述一个地址信息的类。它提供了一组方法来获取地址的各种信息。

程序运行效果如图 12.13 所示。



图 12.13　地理查询

12.6.2　逆地理查询

逆地理查询的过程与地理查询相反，是根据地址名称查询坐标的过程，代码如下：

```
public class MyWebView extends MapActivity {
    private MapView map;                              //定义 MapView 控件
    private GeoPoint center;                          //定义地图中心点
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);                //加载 XML 文件
        map = (MapView) findViewById(R.id.map);       //实例化 MapView 控件
        map.setBuiltInZoomControls(true);             //设置内置缩放工具可用
        //实例化地图中心 GeoPoint 对象
        center = new GeoPoint((int) (39.9067452 * 1E6), (int) (116.391177
        * 1E6));
        final MapController mcontrol = map.getController();
                                                     //获取 MapController 对象
        mcontrol.setCenter(center);                   //设置地图中心位置
        mcontrol.setZoom(10);                         //设置地图缩放级别
        getLocationByAddress();
    }

    protected boolean isRouteDisplayed() {
        return false;
    }

    public void getLocationByAddress(){
```

```
        Geocoder gc = new Geocoder(this, Locale.CHINA);
                                                  //实例化 Geocoder 对象
        try{
            //根据地址名称获取地址 Address 对象集合
            List<Address> address =gc.getFromLocationName("人大会堂西路", 1);
            StringBuilder sb = new StringBuilder();
            if (address.size() > 0){
             Address adsLocation = address.get(0);
                                                  //获取集合中第一个地址对象
             sb.append(adsLocation.getLatitude()).append("\n");
                                                          //获取纬度值
             sb.append(adsLocation.getLongitude());      //获取经度值
           }
            Toast.makeText(MyWebView.this, sb.toString(),Toast.LENGTH_
            LONG).show();
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

代码说明：

Geocoder 类的 getFromLocationName()方法会根据地址名称返回一个结果集合。该方法有两种重载形势，本例中使用的方法需要两个参数，第 1 个是 String 类型，定义地址名称；第 2 个是 int 类型，定义返回结果的个数。

程序运行效果如图 12.14 所示。

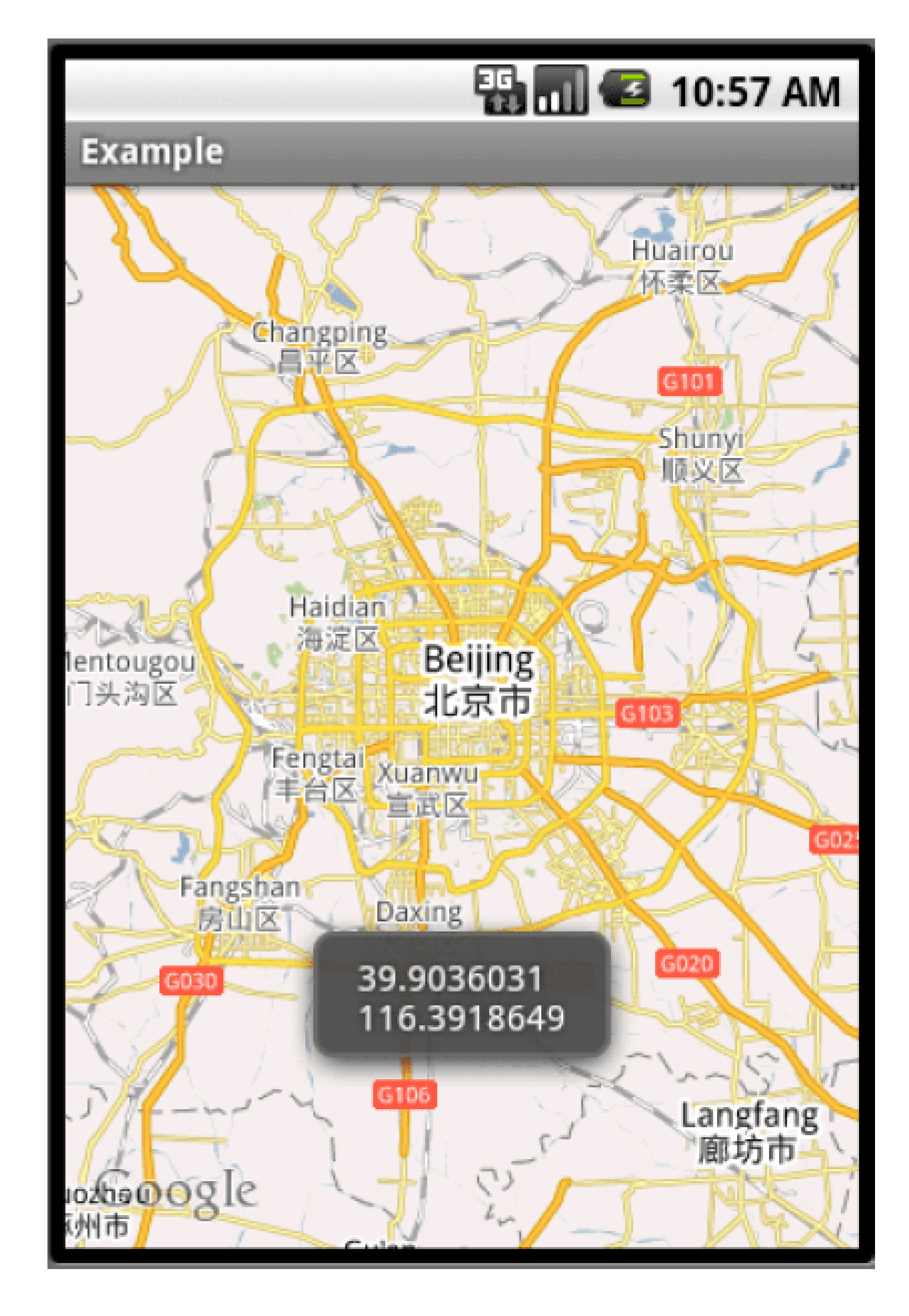


图 12.14　逆地理查询

12.7　在地图上描绘线段

在电子地图上描绘线段，是另一个常见的应用，通常是用来描绘两点之间道路。Android SDK 允许开发者根据起始点的经纬度坐标，调用一个 Activity 来显示两点间的线路，代码如下：

```
Intent intent = new Intent();
intent.setAction(android.content.Intent.ACTION_VIEW);
intent.setData(Uri.parse("http:
                    //maps.google.com/maps?f=d&saddr=起点纬度,起点经度&
                                          daddr=终点纬度,终点经度&hl=cn"));
startActivity(intent);
```

当调用这 Activity 后，程序会切换到一个系统定制的界面，从而脱离了原应用程序的控制。如果希望在原程序中的 MapView 上直接描绘线路，就需要做稍微复杂一些的工作。

和地图标注一样，地图上的线段也需要在地图上附加一个 Overlay，然后在上面描绘线段。下面的例子是在地图上绘制一段两点间的道路线段，基本的流程是从服务器端获取一组坐标点，然后逐一在相邻的两个坐标点之间描绘线段，最后组成一条符合实际道路走向的线段。

首先是从服务器端获取坐标点集合的代码：

```
public class ConnectWeb {
    public List<GeoPoint> getPointList() {
        List<GeoPoint> clist = null;                //定义返回的坐标点集合
        try {
            String url = "http://192.168.1.8:8080/AndroidWeb/LineServlet";
                                                    //服务器地址
            HttpPost request = new HttpPost(url);   //根据地址实例化 URL 对象
            HttpResponse response = new DefaultHttpClient().execute
            (request);                              //生成 HttpResponse 对象
            if (response.getStatusLine().getStatusCode() == HttpStatus.SC_
            OK) {                                   //判断连接是否正常
                String str = EntityUtils.toString(response.getEntity());
                                                    //获取返回字符串
                clist = getList(str);               //解析字符串获取坐标集合
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return clist;
    }

    private List<GeoPoint> getList(String str) {
        List<GeoPoint> clist = new ArrayList<GeoPoint>();
        try {
            JSONArray jay = new JSONArray(str);//将字符串信息转化成 JSON 数组
            for (int i = 0; i < jay.length(); i += 1) {
                JSONObject temp = (JSONObject) jay.get(i);
                                            //将数组中的对象转化成为 JSON 对象
                //获取 JSON 对象数值实例化 GeoPoint 对象
                GeoPoint point = new GeoPoint((int) (Double.valueOf(temp
                        .getString("lat")) * 1E6), (int) (Double.valueOf(temp
                        .getString("lng")) * 1E6));
```

```
                clist.add(point);          //添加进返回集合
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return clist;
    }
}
```

在获得坐标集合后，后续的工作就是根据这些坐标来描绘线段，代码如下：

```
public class MyWebView extends MapActivity {
    private MapView map;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);              //加载 XML 文件
        map = (MapView) findViewById(R.id.map);     //实例化 MapView 控件
        map.setBuiltInZoomControls(true);           //设置内置缩放工具可用
        //实例化地图中心 GeoPoint 对象
        center = new GeoPoint((int) (39.9067452 * 1E6), (int) (116.391177
        * 1E6));
        final MapController mcontrol = map.getController();
                                                //获取 MapController 对象
        mcontrol.setCenter(center);             //设置地图中心位置
        mcontrol.setZoom(10);                   //设置地图缩放级别

        List<GeoPoint> points=new ConnectWeb().getPointList();
                                                //获取坐标集合
        Line line = new Line(points);           //实例化 Line 对象
        map.getOverlays().add(line);            //添加 Overlay 到地图
        map.invalidate();                       //刷新地图
    }

    protected boolean isRouteDisplayed() {
        return false;
    }

    public class Line extends Overlay {
        private List<GeoPoint> points;
        private Paint paint;
        public Line(List<GeoPoint> points) {    //定义线段构造方法
            this.points = points;
            paint = new Paint();                //定义画笔
            paint.setARGB(250, 0, 0, 200);      //定义线段颜色
            paint.setAntiAlias(true);           //消除线段锯齿
            paint.setStyle(Paint.Style.FILL AND STROKE);  //设置线段风格
            paint.setStrokeWidth(4);            //设置线段宽度
        }

        public void draw(Canvas canvas, MapView mapView, boolean shadow) {
            if (!shadow) {
                Projection projection = mapView.getProjection();
                                                //获取 Projection 对象
                if (points != null) {           //判断是否有坐标数据
                    if (points.size() >= 2) {   //判断坐标数据是否多于一个
                        //将前点转换地理坐标为屏幕坐标
                        Point start=projection.toPixels(points.get(0),null);
                        for (int i = 1; i < points.size(); i++) {
```

```
                                        //将后点转换地理坐标为屏幕坐标
                                        Point end = projection.toPixels(points.get(i),
                                        null);
                                        canvas.drawLine(start.x, start.y, end.x, end.y,
                                        paint);              //两点间画线
                                        start = end;         //替换前后点
                                    }
                                }
                            }
                        }
                    }
                }
}
```

代码说明：

- 在地图上按实际地理坐标画线，同样需要将地理坐标转化为屏幕坐标。
- Canvas 类的 drawLine()方法是在两个屏幕坐标之间画一条直线，需要 5 个参数，分别是前点的 x/y 坐标、后点的 x/y 坐标、和一个 Paint 对象。

程序运行效果如图 12.15 所示。

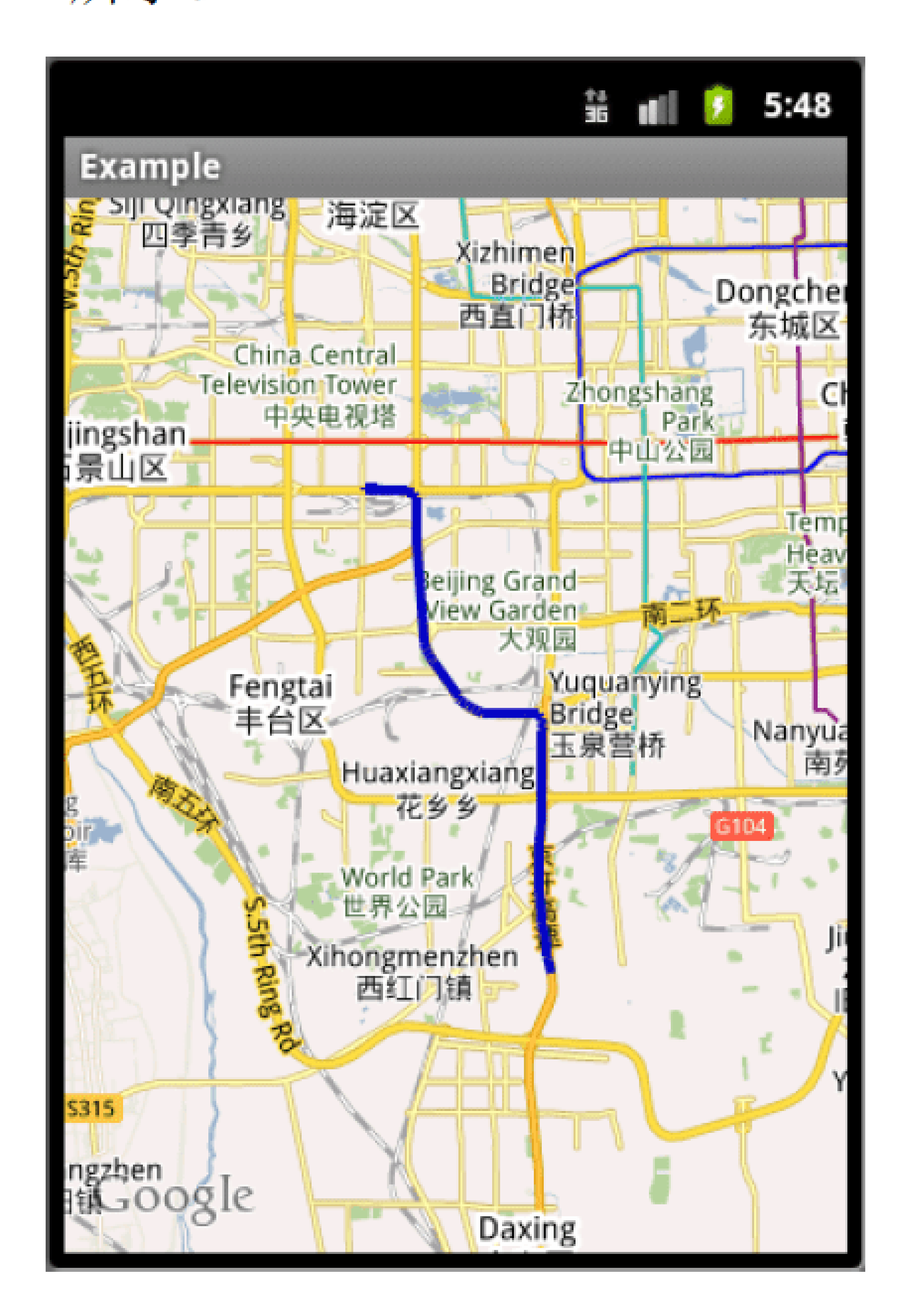


图 12.15　地图上描绘线段

第 3 篇　Android 项目案例实战

⏭　第 13 章　Android 地图定位搜索应用——天涯海角旅游网

⏭　第 14 章　乐乐网上购物商城——边走边购物

第 13 章　Android 地图定位搜索应用——天涯海角旅游网

随着手机技术的飞速发展，手机地图应用已经越来越广泛，手机地图使您随时随地了解乘车路线、搜索目标地址，给人们出行带来了极大的方便，并创造了越来越大的市场。在众多 Android 参考书中，关于地图的介绍极少，案例也不够丰富。本案例是一个综合性的案例，涉及详细的对 Google 地图的应用、在地图上画线、添加 marker、为 marker 添加提示信息等，另外还涉及 GPS 技术、播放 MP3、读取 asserts 下文件、打电话、动态窗体布局等，熟练掌握本案例，会让你受益匪浅。

13.1　地图定位搜索应用功能概述

天涯海角旅游网实现以下基本功能模块：

- 展示精品线路列表；
- 展示精品线路详情；
- 展示精品线路兴趣点列表信息；
- 展示精品线路服务站信息；
- 展示精品线路分段路线信息；
- 在地图上展示线路信息和 Poi 信息；
- 展示兴趣点详情信息；
- 展示服务站信息列表；
- 展示服务站信息详情。

本项目通过读取 asserts 下的线路信息、对线路的基本信息进行展示，以及在地图上展示线路走向和兴趣点的信息，可以隐藏某一个兴趣点，可以显示用户当前的位置，可以听关于某一个兴趣点的 MP3 介绍，另外还可以搜索服务区的信息。下面介绍一下该项目的具体流程。

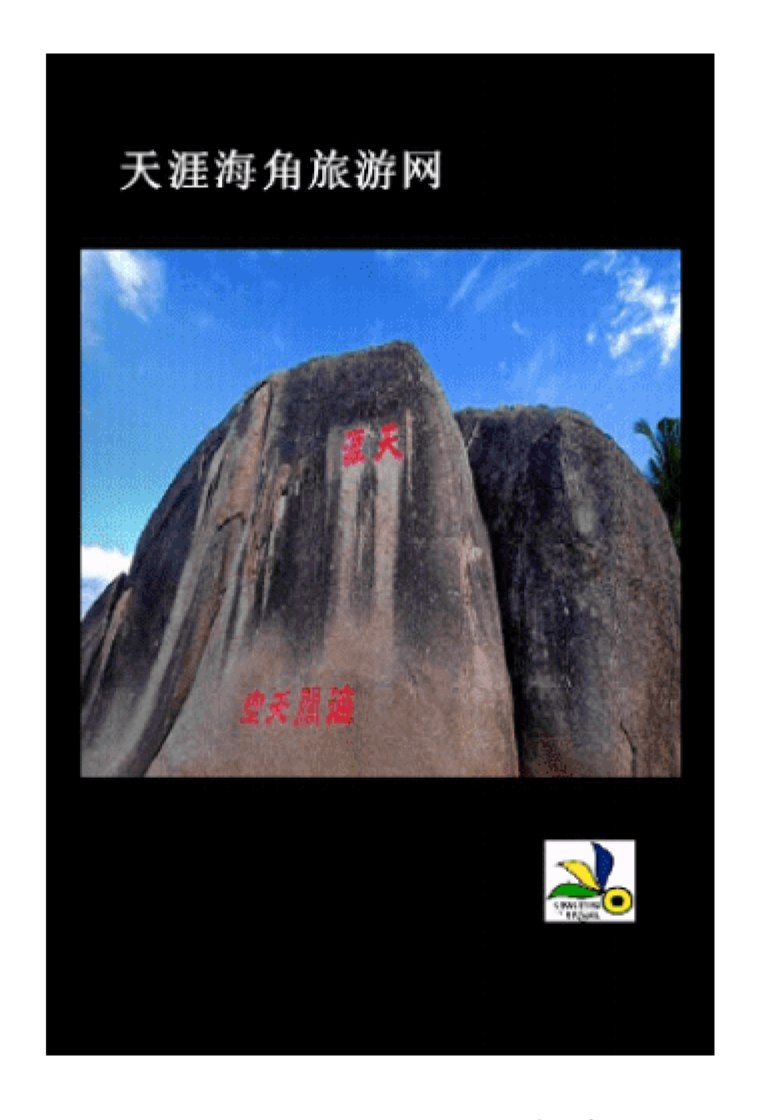
（1）应用启动后，进入欢迎界面，欢迎界面两秒钟后自动跳到 Logo 画面，如图 13.1 和图 13.2 所示。

（2）Logo 画面两秒钟后，跳到应用的主界面，主界面为精品线路列表，如图 13.3 所示。

（3）单击 13.3 主界面中的某一条线路，进入线路详情展示窗体，如图 13.4 和图 13.5 所示。线路详情展示窗体展示了线路的起点、终点、里程、建议行程、线路指数、线路介绍、线路图片等详细信息，另外提供了 3 个 Button，实现对线路“详情”、“兴趣点”、“服务区”窗体的切换。



图 13.1　欢迎界面图



13.2　logo 画面



图 13.3　主界面



图 13.4　线路详情窗体 1



图 13.5　线路详情窗体 2

（4）单击 13.4 中的线路图片，可以展示原图信息，如图 13.6 所示。



图 13.6　线路原图展示

（5）单击图 13.4 中的“详情”按钮，展示线路的分段信息，这里要介绍一下线路的结构，一条线路可以由多条分段线路组成，单击“详情”按钮后的窗体就是展示分段线路的列表，这里模拟的数据中只有一条分段信息，如图 13.7 和图 13.8 所示。

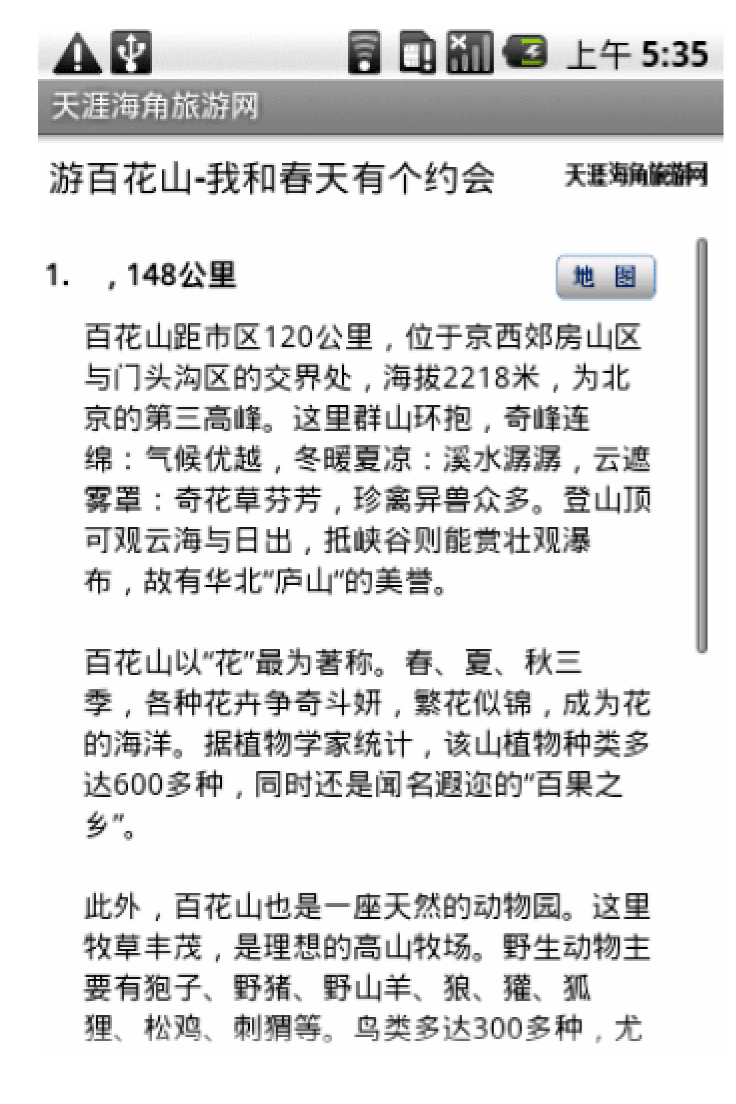


图 13.7　线路分段信息 1



图 13.8　线路分段信息 2

（6）在图 13.7 窗体上展示了线路的分段信息，每一条分段线路都可以在地图上进行展示。单击图 13.7 中的“地图”按钮，会出现地图窗体，并在地图上分段线路上的兴趣点及线路，如图 13.9 所示。也可以通过单击“隐藏兴趣点”选项，隐藏当前地图上显示的兴趣点，如图 13.10 所示。单击“显示当前位置”选项，可以显示用户当前的地理位置；还可以通过单击“返回”选项返回到前一个窗体。另外单击地图上的每一个兴趣点图标后，可以出现该兴趣点的名称提示，如图 13.11 所示。单击该兴趣点的名称提示信息，会跳到兴趣点的详情展示页，如图 13.12 所示。

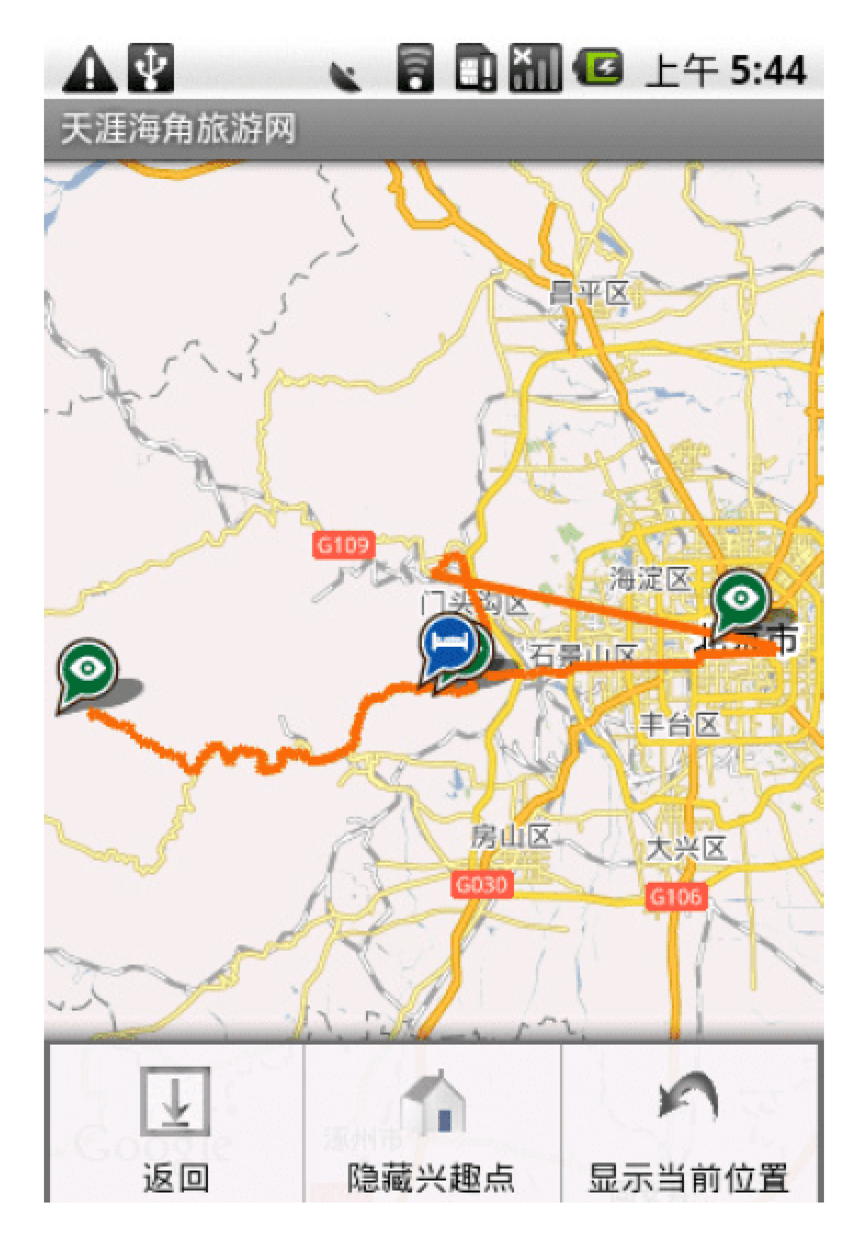


图 13.9　地图窗体

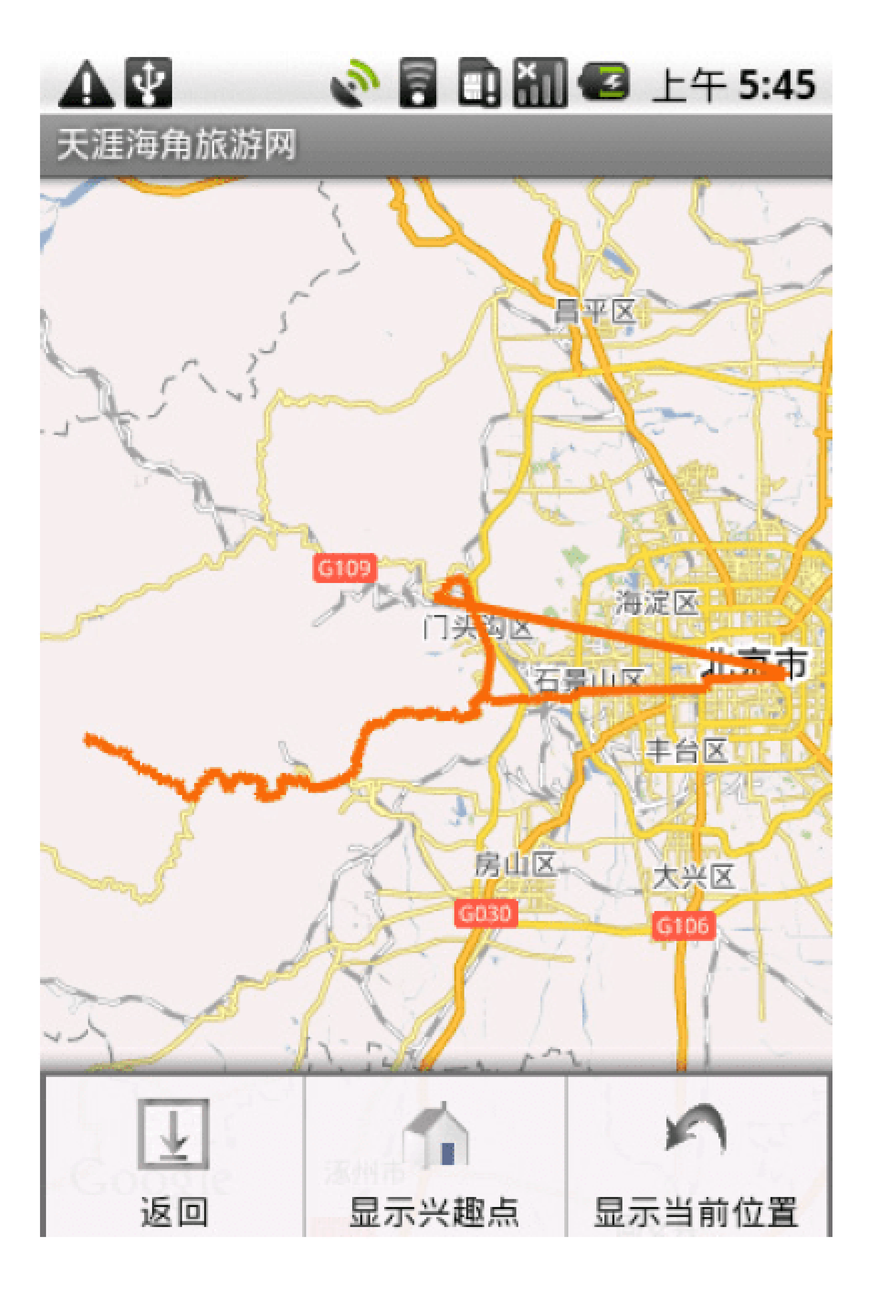


图 13.10　隐藏兴趣点窗体



图 13.11　单击兴趣点图标后效果



图 13.12　兴趣点详情窗体

（7）单击图 13.12 的“带我去”按钮可以获取用户当前位置到兴趣点位置的线路信息，单击“致电”按钮，调用拨号的键盘，用户可以致电兴趣点中的电话，如图 13.13 所示。单击“播放”按钮后开始播放该兴趣点介绍的 MP3，同时“播放”按钮文字变为“停止”，如图 13.14 所示。同样，单击兴趣点详情中的图片，也可以展示兴趣点原图信息。



图 13.13　调用拨号软键盘图



图 13.14　播放 MP3 时，按钮文字变为“停止”

（8）单击图 13.4 中的“兴趣点”按钮，会出现该条分段线路上的兴趣点列表，如图 13.15 所示。



图 13.15　兴趣点列表

（9）单击图 13.15 兴趣点列表中的每一个列表项，会进入兴趣点详情窗体及前面的图 13.12 窗体。

（10）单击图 13.4 中的“服务区”按钮，进入服务区搜索窗体，如图 13.16 所示。在省份的文本框中单击会出现选择省份对话框，如图 13.17 所示。单击“搜索”按钮，按条件进行搜索服务区列表。



图 13.16　服务区搜索窗体



图 13.17　省份选择窗体

13.2　系统包、资源规划的准备工作

13.1 节介绍了项目的流程，在本节中将介绍项目的前期准备工作，包括数据的准备、资源的结构、资源的规划等。本项目在实现的过程中，需要前期准备线路数据资源、图片资源、布局资源等数据信息。线路的数据资源、图片资源、MP3 资源存储在 assets 下，如

图 13.18 所示。

我们用到的线路数据资源文件结构如图 13.18 所示，下面我们来具体讲解资源文件结构。

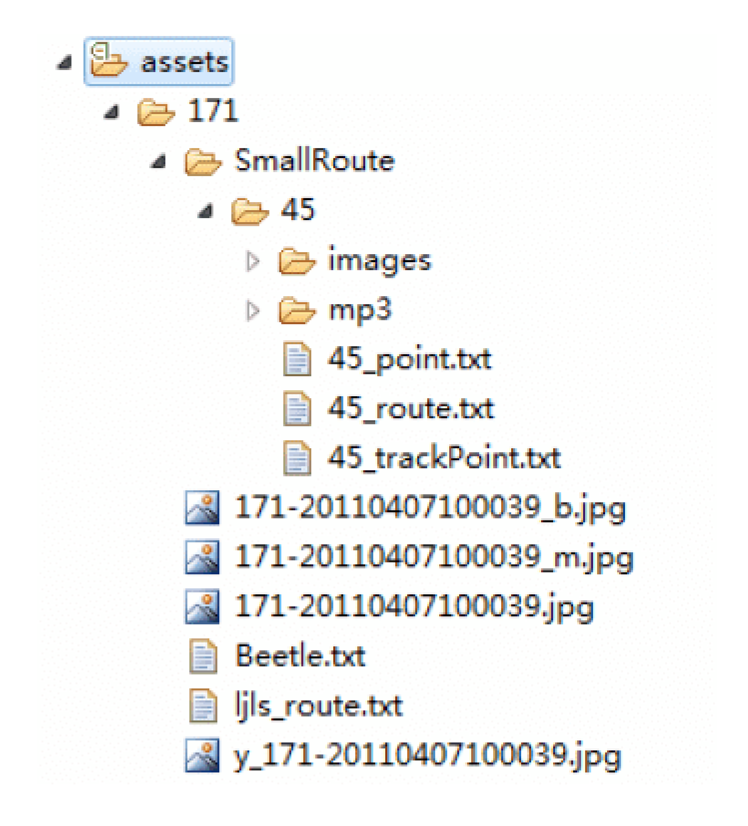


图 13.18　线路数据资源文件

- assets：Android 提供的存放资源文件夹。
- assets /171：整条 ID 为 171 的线路数据文件夹。
- assets /171/SmallRoute：存放线路路段数据文件夹。
- assets /171/SmallRoute /45：一条 ID 为 45 的路段数据文件夹。
- assets /171/SmallRoute /45/images：存放该路段的所属图片文件。
- assets /171/SmallRoute /45/mp3：存放该路段的所属 MP3 文件。
- assets /171/SmallRoute /45/45_point.txt：该路段的兴趣点数据文件。
- assets /171/SmallRoute /45/45_route.txt：该路段的路段数据文件。
- assets /171/SmallRoute /45/45_trackPoint.txt：该路段的线路点数据文件。
- assets /171/*.jpg：为整条 ID 为 171 的线路图片数据。
- assets /171/ljls_route.txt：所有路线用于列表展示的数据。

项目中的 Logo、按钮等图片存储到 res/drawable 位置，项目中的窗体布局文件存储在 res/layout 位置。

13.3　访问资源权限配置

本项目中涉及到读写文件资源、访问 GPS 定位、打电话等功能，这些功能的实现首先需要在 AndroidManifest.xml 中添加允许使用这个功能的权限。另外本项目中窗体均为纵向显示，只有图片详情展示窗体可以横向显示也可以纵向显示，这里设置窗体纵向显示通过设置<activity>标签的属性 android:screenOrientation="portrait"实现。

AndroidManifest.xml 中代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.tyhj" android:versionCode="1" android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/
    app_name">
        <!--欢迎窗体-->
        <activity android:name=".Welcome" android:label="@string/app_name"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <!-- android:screenOrientation="portrait"：强制手机纵向显示 属性值
        "landscape"表示横向显示 -->
        <activity android:name=".Welcome1" android:label="@string/app_name"
            android:screenOrientation="portrait" />
```

```
        <!--线路列表窗体-->
        <activity android:name=".TripList" android:label="@string/app_name"
            android:screenOrientation="portrait" />
        <!--线路详情窗体-->
        <activity android:name=".TripDetail" android:label="@string/app_name"
            android:screenOrientation="portrait" />
        <!--线路分段窗体-->
        <activity android:name=".TripSegment" android:label="@string/app_name"
            android:screenOrientation="portrait" />
        <!--兴趣点列表窗体-->
        <activity android:name=".TripPoiList" android:label="@string/
            app_name"android:screenOrientation="portrait" />
        <!--服务区列表窗体-->
        <activity android:name=".DZDealersList" android:label="@string/
            app_name"android:screenOrientation="portrait" />
        <!--地图窗体-->
        <activity android:name=".RoadMapView" android:label="@string/
            app_name"android:screenOrientation="portrait" />
        <activity android:name=".PoiDetail" android:label="@string/app_name"
            android:screenOrientation="portrait" />
        <!--服务区详情窗体-->
        <activity android:name=".DZDealersDetail" android:label="@string/
            app_name"android:screenOrientation="portrait" />
        <!--图片展示窗体-->
        <activity android:name=".TripDetailPic" android:label="@string/
        app_name" />
        <!--引入google 地图库-->
        <uses-library android:name="com.google.android.maps" />
    </application>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_
    LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.CALL_PHONE" />
</manifest>
```

13.4　项目架构介绍

本项目是在手机端运行的，所有的数据资源均存储在手机端。下面简单介绍一下本项目中各个类的功能。

13.4.1　实体类简要介绍

该应用涉及线路实体类 Route、兴趣点实体类 PoiPoint、MP3 实体类 Mp3Point、线路轨迹实体类 TrackPoint、服务区实体类 Beetle。类基本功能如下。

- 线路类 Route：描述线路的基本信息，提供修改线路信息，获取线路信息的方法。
- 兴趣点类 PoiPoint：描述兴趣点的基本信息，提供修改兴趣点信息，获取兴趣点信息的方法。
- MP3 类 Mp3Point：描述 MP3 的基本信息，提供修改 MP3 信息，获取 MP3 信息的方法。

- 线路轨迹类 TrackPoint：描述线路轨迹的基本信息，提供修改线路轨迹信息，获取线路轨迹信息的方法。
- 服务区类 Beetle：描述服务区的基本信息，提供修改服务区信息，获取服务区信息的方法。

13.4.2　工具类简要介绍

项目实现过程中涉及对本地数据资源进行加密，存储省份信息资源及文件访问操作的相关工具类。下面具体介绍每一个类的功能。

- DESCoder 类：处于对数据的保护，手机端的数据文件均已加密，该类中提供了加密和解密的方法。
- Keyfile 类：定义数据文件密钥。
- StaticString 类：定义省份的数据信息。
- WAnalysisFile 类：负责读取文件中的线路信息、兴趣点信息、MP3 信息、轨迹线信息、服务器信息等。

13.4.3　界面相关类简要介绍

界面相关类的简要介绍如下。

- Welcome 类：欢迎窗体类，欢迎窗体两秒钟后自动跳到 Logo 窗体。
- Welcome1 类：Logo 窗体类，展示网站的 Logo 信息。两秒钟后跳到线路列表窗体。
- TripList 类：线路列表窗体类。展示精品线路信息。通过工具类 WAnalysisFile 中提供的方法，请求线路列表信息，展示在窗体上。
- TripDetail 类：线路详情窗体类。展示线路详情信息。
- TripDetailPic 类：线路图片窗体类。展示线路图片的原图信息。
- TripSegment 类：线路分段窗体类。展示线路的分段信息。通过工具类 WAnalysisFile 中提供的方法，请求线路分段信息，展示在窗体上。
- TripPoiList 类：兴趣点列表窗体类。通过工具类 WAnalysisFile 中提供的方法，请求兴趣点列表信息，展示在窗体上。
- PoiDetail 类：兴趣点详情窗体类。
- DZDealersList 类：服务区窗体类。通过工具类 WAnalysisFile 中提供的方法，请求服务区列表信息，展示在窗体上。
- DZDealersDetail 类：服务区详情窗体类。

13.5　实体类代码实现

上一节中简单介绍了项目中涉及的实体类及功能，实体类就是用来描述一个对象的类，Java 是基于面向对象编程，所以实体类就显得尤为重要了。接下来我们详细看一下各个实体类的代码实现。

13.5.1　线路实体类 Route

Route 类描述线路的基本信息，如线路名称、里程、特别提示、描述等一些信息，值得注意的 pointList 属性和 trackPointList 属性。

pointList 属性是一个集合类型的属性，集合中封装的是 PoiPoint 兴趣点类，表示这条线路的所有兴趣点。trackPointList 属性和 pointList 属性一样，也是一个集合类型的属性，它封装的是 TrackPoint 线路轨迹类，表示这条线路的所有轨迹信息。

Java 代码如下：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class Route implements Serializable {
    private String id = "";                       //编号
    private String name = "";                     //名称
    private String wayPoint = "";                 //途经点名称字符串
    private String startPointName = "";           //起点名称
    private String startPoint = "";               //起点经纬度
    private String endPointName = "";             //终点名称
    private String endPoint = "";                 //终点经纬度
    private String city = "";                     //城市
    private String mileage = "";                  //里程
    private String roadToll = "";                 //路桥费
    private String drivingTime = "";              //驾驶时间
    private String proposedItinerary = "";        //建议行程
    private String bestSeason = "";               //最佳季节
    private int recommend = 0;                    //驾驶指数
    private int scenic = 0;                       //风光指数
    private int renwen = 0;                       //人文指数
    private int food = 0;                         //美食指数
    private String tip = "";                      //特别提示
    private String trend = "";                    //线路走向
    private String image = "";                    //线路图片
    private String highlights = "";               //亮点
    private String drivingTips = "";              //驾驶指引
    private String desc = "";                     //描述
    private List<PoiPoint> pointList = new ArrayList<PoiPoint>();
                                                  //PoiPoint 集合
    private List<TrackPoint> trackPointList = new ArrayList<TrackPoint>();
                                                  //TrackPoint 集合
……//该处省略了成员变量的 getXXX()和 setXXX()方法的代码，读者可自行查阅随书光盘中的
    源代码
}
```

13.5.2　兴趣点实体类 PoiPoint

PoiPoint 类描述兴趣点的基本信息，值得一讲的是这个类中也有一个集合类型的属性 imgList。该属性是一个 String 类的集合，里面存放的是图片文件的名称，因为一个兴趣点有可能是一张或者多张图片，所以这里要用一个集合来描述。

Java 代码如下：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class PoiPoint implements Serializable{
    private String id = "";                          //编号
    private String name = "";                        //名称
    private String routeId = "";                     //路书编号
    private String time = "";                        //时间
    private String lat = "";                         //纬度
    private String lon = "";                         //经度
    private String categoryId = "";                  //类型编号
    private String mp3Id = "";                       //mp3 名称
    private String mp3Path = "";                     //mp3 全名
    private int mp3Range=0;                          //范围
    private String tel="";                           //电话
    private String desc="";                          //描述
    private String address="";                       //地址
    private List<String> imgList = new ArrayList<String>();   //图片集合

    ……//该处省略了成员变量的 getXXX()和 setXXX()方法的代码，读者可自行查阅随书光盘
        中的源代码
}
```

13.5.3　MP3 实体类 Mp3Point

Mp3Point 类描述 MP3 的基本信息，lat 属性是描述的纬度，lon 属性是描述的经度，看到这里一定很奇怪了，为什么 MP3 会有经纬度描述，因为这个类主要应用于手机地图上，比如靠近某个兴趣点的一定范围，自动播放 MP3，所以在 Mp3Point 中定义经纬度及 mp3Range 属性（范围描述）就很有必要了。

Java 代码如下：

```
package com.tyhj;
public class Mp3Point {
    private String lat = "";                         //纬度
    private String lon = "";                         //经度
    private String mp3Id = "";                       //编号
    private String mp3Path = "";                     //路径
    private boolean pan=true;
    private int mp3Range=0;                          //范围
    ……//该处省略了成员变量的 getXXX()和 setXXX()方法的代码，读者可自行查阅随书光盘
        中的源代码
}
```

13.5.4　线路轨迹实体类 TrackPoint

TrackPoint 类描述线路轨迹的基本信息，其中 routeId 属性是一个关联属性，它关联 Route 类中的 id 属性，能更好匹配。trackPoints 属性是一个 String 字符串类型，它存放这一段轨迹的轨迹点，是 TrackPoint 类的核心属性。

Java 代码如下：

```
package com.tyhj;
public class TrackPoint {
    private String id = "";                     //编号
    private String name = "";                   //名称
    private String routeId = "";                //route 编号
    private String desc = "";                   //描述
    private String time = "";                   //时间
    private String lat = "";                    //维度
    private String lon = "";                    //经度
    private String categoryId = "";             //类型
    private String trackPoints="";              //经纬度字符串
    ……//该处省略了成员变量的 getXXX()和 setXXX()方法的代码，读者可自行查阅随书光盘
        中的源代码
}
```

13.5.5　服务区实体类 Beetle

Beetle 类描述服务区的基本信息，如公司名称、维度、经度、公司地址等一系列信息，这里的经纬度描述主要用于在地图上标注商家所在的位置。

Java 代码如下：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class Beetle implements Serializable {
    private String id = "";                     //编号
    private String name = "";                   //公司名称
    private String lat = "";                    //维度
    private String lon = "";                    //经度
    private String contacts = "";               //公司联系方式
    private String tel = "";                    //公司电话
    private String address = "";                //公司地址
    private String zip = "";                    //公司邮编
    private String fax = "";                    //公司传真
    private String email = "";                  //公司邮箱
    ……//该处省略了成员变量的 getXXX()和 setXXX()方法的代码，读者可自行查阅随书光盘
        中的源代码
}
```

13.6　加密工具类代码实现

13.6.1　加密工具类 DESCoder

DESCoder 类是一个加密工具类，主要是 DES 加密原理，有兴趣的朋友可自己去翻阅一些资料，这里就不详解了。

DESCoder 类主要有 3 个方法，即 toKey()转换密钥方法、encrypt()加密方法和 decrypt()解密方法。toKey()转换密钥方法主要是内部使用，在加密和解密时调用。encrypt()加密方法有两个参数，第 1 个 byte[]数组类型参数，表示要加密的 byte[]数组；第 2 个 String 类型

参数，表示自定义密钥。decrypt()解密方法和 encrypt()加密方法的参数一样，也是一个需要解密的 byte[]数组和一个自定义密钥。

Java 代码如下：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public abstract class DESCoder{
    public static final String ALGORITHM = "DES";

    /**
     * 转换密钥
     *
     * @param key
     * @return
     * @throws Exception
     */
    private static Key toKey(byte[] key) throws Exception {
        DESKeySpec dks = new DESKeySpec(key);
        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance
        (ALGORITHM);
        SecretKey secretKey = keyFactory.generateSecret(dks);
        //当使用其他对称加密算法时，如 AES、Blowfish 等算法时，用下述代码替换上述三
          行代码
        //SecretKey secretKey = new SecretKeySpec(key, ALGORITHM);
        return secretKey;
    }

    /**
     * 解密
     *
     * @param data
     * @param key
     * @return
     * @throws Exception
     */
    public static byte[] decrypt(byte[] data, String key) throws Exception {
//      Key k = toKey(decryptBASE64(key));
        Key k = toKey(key.getBytes());
        Cipher cipher = Cipher.getInstance(ALGORITHM);
        cipher.init(Cipher.DECRYPT_MODE, k);

        return cipher.doFinal(data);
    }

    /**
     * 加密
     *
     * @param data
     * @param key
     * @return
     * @throws Exception
     */
    public static byte[] encrypt(byte[] data, String key) throws Exception {
        Key k = toKey(key.getBytes());
        Cipher cipher = Cipher.getInstance(ALGORITHM);
        cipher.init(Cipher.ENCRYPT_MODE, k);
        return cipher.doFinal(data);
    }
}
```

13.6.2　定义数据文件密钥类 Keyfile

Keyfile 类主要定义了一个自定义密钥，定义了一个公共的静态常量 ROAD_BOOK_KEY，方便程序的维护和扩展。

Java 代码如下：

```
package com.tyhj;

public class Keyfile {
    //数据文件密钥
    public static final String ROAD_BOOK_KEY = "RoadBook难￥的@#答案□?";
}
```

13.7　文件访问工具类代码实现

手机端窗体显示的数据信息，都是从文件中读取，在 WAnalysisFile 类中提供了丰富的对文件读取的方法。例如，读取文件并解密的方法、获取 Route 对象的方法、获取 Route 集合的方法、获取 PoiPoint 对象的方法等。

WAnalysisFile 类主要目的是保证整个程序的数据，程序所有用到的数据，都是通过该类获取的，是程序必不可缺的核心类。

WAnalysisFile 类的原理是基于对数据文件的解析，通过对数据文件的解析，能让我们获取想要的数据，除了对数据的解析，涉及最多的就是对集合的操作，对集合的增删改查、排序、去重等一系列操作。具体的方法和作用，在代码中都有具体的注释，下面让我们来看具体代码。

Java 代码如下：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class WAnalysisFile {
    /**
     * 读取文件并解密
     *
     * @param context
     * @param path
     * @return
     */
    public String readFileByByte(Context context, String path) {
        try {
            InputStream iStream = context.getAssets().open(path);
                                                      //获取文件路径
            byte[] b = new byte[iStream.available()];  //初始化 byte 数组
            iStream.read(b);                          //读取文件到 byte[]
            iStream.close();                          //关闭文件流
            b = DESCoder.decrypt(b, Keyfile.ROAD_BOOK_KEY);    //解密
            String content = new String(b);           //byte 数组转 String
            return content;
        } catch (Exception e) {
```

```
            e.printStackTrace();
            return "";
        }
    }

    /**
     * 获取 route 对象，根据 routeId
     *
     * @param context
     * @param routeId
     * @return
     */
    public Route getRouteById(Context context, String routeId) {
        String path = "SmallRoute/" + routeId + "/" + routeId + "_route.txt";
                                        //获取文件路径
        String content = readFileByByte(context, path);
                                        //调用文件读取方法，获取字符串
        if (content != null && !"".equals(content)) {
                                        //判断文件字符是否为 NULL 或空字符
            return setRoute(content);   //调用解析字符串方法，获取 Route 对象
        } else {
            return new Route();         //返回默认 Route 对象
        }
    }

    /**
     * 获取 route 对象，根据 bigRouteId,routeId
     *
     * @param context
     * @param bigRouteId
     * @param routeId
     * @return
     */
    public Route getRouteById(Context context, String bigRouteId, String
    routeId) {
        String path = bigRouteId + "/SmallRoute/" + routeId + "/" + routeId
                + "_route.txt";         //获取文件路径
        String content = readFileByByte(context, path);
                                        //调用文件读取方法，获取字符串
        if (content != null && !"".equals(content)) {
                                        //判断文件字符是否为 NULL 或空字符
            return setRoute(content);   //调用解析字符串方法，获取 Route 对象
        } else {
            return new Route();         //返回默认 Route 对象
        }
    }

    /**
     * 获取 Route 集合
     *
     * @param context
     * @return
     */
    public List<Route> getRouteList(Context context) {
        List<Route> routeList = new ArrayList<Route>();//初始化 Route 集合
        try {
            String[] routes = context.getAssets().list("SmallRoute");
```

```
                                            //获取 Route 在资源文件夹的名称数组
            for (int i = 0; i < routes.length; i++) {
                Route route = getRouteById(context, routes[i]);
                                            //调用根据 routeId 获取 Route 对象方法
                routeList.add(route);       //添加 Route 到 Route 集合
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return routeList;                   //返回 Route 集合
    }

    /**
     * 获取 Route 集合，根据 bigRouteId
     *
     * @param context
     * @param bigRouteId
     * @return
     */
    public List<Route> getRouteList(Context context, String bigRouteId) {
        List<Route> routeList = new ArrayList<Route>();//初始化 Route 集合
        try {
            String[] routes = context.getAssets().list(
                    bigRouteId + "/SmallRoute");
                                            //获取 Route 在资源文件夹的名称数组
            for (int i = 0; i < routes.length; i++) {
                Route route = getRouteById(context, bigRouteId, routes[i]);
                                //调用根据 bigRouteId，routeId 获取 Route 对象方法
                routeList.add(route);       //添加 Route 到 Route 集合
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return routeList;                   //返回 Route 集合
    }

    /**
     * 填充 Route 对象
     *
     * @param content
     * @return
     */
    public Route setRoute(String content) {
        String[] routes = content.split("@#@");
                                        //分割字符串，获取 Route 相应的字符串数组
        Route route = new Route(); //初始化 Route 对象
        if (routes.length > 0) {
            String[] routeProperty = routes[0].split("!#!");
                                        //分割字符串，获取 Route 相应的字符串数组
            route.setId(routeProperty[0].trim());            //设置 Id
            route.setName(routeProperty[1].trim());          //设置 Name
            route.setWayPoint(routeProperty[2].trim());      //设置 WayPoint
            route.setStartPointName(routeProperty[3].trim());
                                        //设置 StartPointName
            route.setStartPoint(routeProperty[4].trim());//设置 StartPoint
            route.setEndPointName(routeProperty[5].trim());//设置 EndPointName
            route.setEndPoint(routeProperty[6].trim()); //设置 EndPoint
```

```
            route.setCity(routeProperty[7].trim());     //设置 City
            route.setMileage(routeProperty[8].trim()); //设置 Mileage
            route.setRoadToll(routeProperty[9].trim());//设置 RoadToll
            route.setDrivingTime(routeProperty[10].trim());//设置 DrivingTime
            route.setProposedItinerary(routeProperty[11].trim());
                                               //设置 ProposedItinerary
            route.setBestSeason(routeProperty[12].trim());//设置 BestSeason
            route.setRecommend(Integer.parseInt(routeProperty[13].trim()));
                                                       //设置 Recommend
            route.setScenic(Integer.parseInt(routeProperty[13].trim()));
                                                       //设置 Scenic
            route.setRenwen(Integer.parseInt(routeProperty[15].trim()));
                                                       //设置 Renwen
            route.setFood(Integer.parseInt(routeProperty[16].trim()));
                                                       //设置 Food
            route.setTip(routeProperty[17].trim());     //设置 Tip
            route.setTrend(routeProperty[18].trim());  //设置 Trend
            route.setImage(routeProperty[19].trim().toLowerCase());
                                                       //设置 Image
            route.setHighlights(routeProperty[20].trim());//设置 Highlights
            route.setDrivingTips(routeProperty[21].trim());//设置 DrivingTips
            if (routeProperty.length >= 23) {
                route.setDesc(routeProperty[22].trim());//设置 Desc
            }
        }
        return route;//返回 Route 对象
    }

    /**
     * 填充 PoiPoint 对象
     *
     * @param pointProperty
     * @return
     */
    public PoiPoint setPoint(String[] pointProperty) {
        PoiPoint point = new PoiPoint();             //初始化 PoiPoint 对象
        point.setId(pointProperty[0].trim());          //设置 Id
        point.setName(pointProperty[1].trim());        //设置 Name
        point.setRouteId(pointProperty[2].trim());     //设置 RouteId
        point.setMp3Id(pointProperty[3].trim());       //设置 Mp3Id
        point.setMp3Path(pointProperty[4].trim());     //设置 Mp3Path
        point.setLat(pointProperty[5].trim());         //设置 Lat
        point.setLon(pointProperty[6].trim());         //设置 Lon
        point.setCategoryId(pointProperty[7].trim()); //设置 CategoryId
        String imgString = pointProperty[8].trim();    //获取 img 字符
        //img
        if (!"".equals(imgString)) {
            List<String> imgList = new ArrayList<String>();//初始化 img 集合
            String[] imgs = imgString.split("[|]");        //获取 img 数组
            for (int j = 0; j < imgs.length; j++) {
                if (!"".equals(imgs[j])) {
                    imgList.add(imgs[j].toLowerCase());//将 img 添加到 img 集合
                }
            }
            point.setImgList(imgList);                   //设置 ImgList
```

```
        }
        if ("null".equals(pointProperty[9].trim())) {
            point.setDesc("");                          //设置默认 Desc
        } else {
            point.setDesc(pointProperty[9].trim());     //设置 Desc
        }
        point.setTel(pointProperty[10].trim());         //设置 Tel
        point.setAddress(pointProperty[11].trim());     //设置 Address
        point.setMp3Range(Integer.parseInt(pointProperty[12].trim()));
                                                        //设置 Mp3Range
        if (pointProperty.length >= 13) {
            point.setTime(pointProperty[13].trim());    //设置 Time
        }
        return point;                                   //返回 PoiPoint 对象
    }

    /**
     * 获取 PoiPoint 对象，根据 PoiPoint 的 id
     *
     * @param content
     * @param pointId
     * @return
     */
    public PoiPoint getPointById(String content, String pointId) {
        String[] points = content.split("@#@");     //分割字符串
        PoiPoint point = new PoiPoint();            //初始化 PoiPoint 对象
        if (points.length > 0) {
            for (int i = 0; i < points.length; i++) {
                String[] pointProperty = points[i].split("!#!");
                                    //分割字符串，获取相应的字符串数组
                if (pointProperty[0].equals(pointId)) {
                    point = setPoint(pointProperty);
                                    //调用填充 PoiPoint 对象方法
                }
            }
        }
        return point;                       //返回 PoiPoint 对象
    }

    /**
     * 获取大路书详情
     *
     * @param context
     * @return
     */
    public Route getBigRoute(Context context) {
        String path = "ljls route.txt";                 //获取路径
        String content = readFileByByte(context, path);
                                  //调用文件读取方法，获取字符串
        if (content != null && !"".equals(content)) {
            return setRoute(content);//调用填充 Route 对象方法，返回 Route 对象
        } else {
            return new Route();      //返回默认的 Route 对象
        }
    }

    /**
```

```
     * 获取大路书详情根据 bigRouteId
     *
     * @param context
     * @param bigRouteId
     * @return
     */
    public Route getBigRoute(Context context, String bigRouteId) {
        String path = bigRouteId + "/ljls_route.txt";      //获取路径
        String content = readFileByByte(context, path);
                                    //调用文件读取方法，获取字符串
        if (content != null && !"".equals(content)) {
            Route route = setRoute(content);
                                    //调用填充 Route 对象方法，获取 Route 对象
            return route;           //返回 Route 对象
        } else {
            return new Route();     //返回默认的 Route 对象
        }
    }

    /**
     * 获取大路书集合
     *
     * @param context
     * @return
     */
    public List<Route> getBigRouteList(Context context) {
        List<Route> routeList = new ArrayList<Route>();//初始化 Route 集合
        try {
            String[] routeDirs = context.getAssets().list("");
                                            //获取文件的名称数组
            for (int i = 0; i < routeDirs.length; i++) {
                Route route = new Route();      //初始化 Route 对象
                String path = routeDirs[i] + "/ljls route.txt";
                                            //获取文件路径
                String content = readFileByByte(context, path);
                                            //调用文件读取方法，获取字符串
                if (content != null && !"".equals(content)) {
                    route = setRoute(content);
                                    //调用填充 Route 对象方法，获取 Route 对象
                    route.setId(routeDirs[i]); //设置 Id
                    routeList.add(route);       //添加 Route 对象到 Route 集合
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return routeList;                   //返回 Route 集合
    }

    /**
     * 获取每条 route 的 PoiPoint 集合
     *
     * @param context
     * @param routeId
     * @return
     */
    public List<PoiPoint>getSmallPoiPointList(Context context,String routeId) {
```

```
    String path = "SmallRoute/" + routeId + "/" + routeId + " point.txt";
                                              //获取文件路径
    String content = readFileByByte(context, path);
                                          //调用文件读取方法，获取字符串
    List<PoiPoint> pointList = new ArrayList<PoiPoint>();
                                          //初始化 PoiPoint 集合
    if (content != null && !"".equals(content)) {
        String[] points = content.split("@#@");     //分割字符串
        if (points.length > 0) {
            for (int i = 0; i < points.length; i++) {
                String[] pointProperty = points[i].split("!#!");
                                                        //分割字符串
                //调用填充 PoiPoint 对象方法，获取 PoiPoint 对象
                PoiPoint point = setPoint(pointProperty);
                //添加 PoiPoint 对象到 PoiPoint 集合
                pointList.add(point);
            }
        }
    }
    return pointList;                                   //返回 PoiPoint 集合
}

/**
 * 获取每条 route 的 PoiPoint 集合
 *
 * @param context
 * @param routeId
 * @return
 */
public List<PoiPoint> getSmallPoiPointList(Context context, String routeId,
        String bigRouteId) {
    String path = bigRouteId + "/SmallRoute/" + routeId + "/" + routeId
            + "_point.txt";                             //获取文件路径
    String content = readFileByByte(context, path);
                                              //调用文件读取方法，获取字符串
    List<PoiPoint> pointList = new ArrayList<PoiPoint>();
                                              //初始化 PoiPoint 集合
    if (content != null && !"".equals(content)) {
        String[] points = content.split("@#@");     //分割字符串
        if (points.length > 0) {
            for (int i = 0; i < points.length; i++) {
                String[] pointProperty = points[i].split("!#!");
                                                        //分割字符串
                //调用填充 PoiPoint 对象方法，获取 PoiPoint 对象
                PoiPoint point = setPoint(pointProperty);
                //添加 PoiPoint 对象到 PoiPoint 集合
                pointList.add(point);
            }
        }
    }
    return pointList;                                   //返回 PoiPoint 集合
}

/**
 * 得到大路书的 PoiPoint 集合
 *
 * @param context
 * @return
```

```
     */
    public List<PoiPoint> getBigPoiPointList(Context context) {
        try {
            List<PoiPoint> pointList = new ArrayList<PoiPoint>();
                                                    //初始化 PoiPoint 集合
            String[] routes = context.getAssets().list("SmallRoute");
                                                    //获取文件名称数组
            for (int i = 0; i < routes.length; i++) {
                //获取每条 route 的 PoiPoint 集合
                List<PoiPoint> pointSmallList = getSmallPoiPointList(context,
                        routes[i]);
                for (int j = 0; j < pointSmallList.size(); j++) {
                    //添加 PoiPoint 对象到 PoiPoint 集合
                    pointList.add(pointSmallList.get(j));
                }
            }
            //调用去重方法，去掉重复的 PoiPoint 对象
            pointList = removeRepeatObject(pointList);
            return pointList;                       //返回 PoiPoint 集合
        } catch (Exception e) {
            e.printStackTrace();
            return new ArrayList<PoiPoint>();   //返回默认 PoiPoint 集合
        }
    }

    /**
     * 得到大路书的 PoiPoint 集合
     *
     * @param context
     * @return
     */
    public List<PoiPoint>getBigPoiPointList(Context context,String bigRouteId){
        try {
            List<PoiPoint> pointList = new ArrayList<PoiPoint>();
                                                        //初始化 PoiPoint 集合
            String[] routes = context.getAssets().list(
                    bigRouteId + "/SmallRoute");    //获取文件名称数组
            for (int i = 0; i < routes.length; i++) {
                List<PoiPoint> pointSmallList = getSmallPoiPointList(context,
                        routes[i],bigRouteId);//获取每条 route 的 PoiPoint 集合
                for (int j = 0; j < pointSmallList.size(); j++) {
                    //添加 PoiPoint 对象到 PoiPoint 集合
                    pointList.add(pointSmallList.get(j));
                }
            }
            //调用去重方法，去掉重复的 PoiPoint 对象
            pointList = removeRepeatObject(pointList);
            return pointList;                       //返回 PoiPoint 集合
        } catch (Exception e) {
            e.printStackTrace();
            return new ArrayList<PoiPoint>();   //返回默认 PoiPoint 集合
        }
    }

    /**
     * 去重复
     *
     * @param pointList
```

```
     * @return
     */
    public List<PoiPoint> removeRepeatObject(List<PoiPoint> pointList) {
        ……//该处省略了具体实现代码，将在之后进行具体介绍
    }

    /**
     * 根据 route 的 id 查询该 route 的所有 TrackPoint
     *
     * @param routeId
     * @return
     */
    public List<TrackPoint> getTrackPointListByRouteId(Context context,
            String routeId) {
        ……//该处省略了具体实现代码，将在之后进行具体介绍
    }

    /**
     * 根据 route 的 id 查询该 route 的所有 TrackPoint
     *
     * @param routeId
     * @return
     */
    public List<TrackPoint> getTrackPointListByRouteId(Context context,
            String routeId, String bigRouteId) {
        String path = bigRouteId + "/SmallRoute/" + routeId + "/" + routeId
                + "_trackPoint.txt";            //获取文件路径
        //调用文件读取方法，获取字符串
        String content = readFileByByte(context, path);
        if (content != null && !"".equals(content)) {
        //调用填充 TrackPoint 集合，并返回 TrackPoint 集合
            return setTrackPointList(content);
        } else {
            return new ArrayList<TrackPoint>();//返回默认 TrackPoint 集合
        }
    }

    /**
     * 填充 TrackPoint 集合
     *
     * @param content
     * @return
     */
    public List<TrackPoint> setTrackPointList(String content) {
        String[] trackPoints = content.split("@#@");   //分割字符串
        List<TrackPoint> trackPointList = new ArrayList<TrackPoint>();
                                        //初始化 TrackPoint 集合
        if (trackPoints.length > 0) {
            for (int i = 0; i < trackPoints.length; i++) {
                String[] trackPointProperty = trackPoints[i].split("!#!");
                                        //分割字符串
                TrackPoint trackPoint = new TrackPoint();
                                        //初始化 TrackPoint 对象
                trackPoint.setId(trackPointProperty[0].trim());//设置 Id
                trackPoint.setName(trackPointProperty[1].trim());//设置 Name
                trackPoint.setRouteId(trackPointProperty[2].trim());
                                        //设置 RouteId
```

```
                trackPoint.setDesc(trackPointProperty[3].trim());//设置 Desc
                trackPoint.setLat(trackPointProperty[4].trim()); //设置 Lat
                trackPoint.setLon(trackPointProperty[5].trim()); //设置 Lon
                trackPoint.setCategoryId(trackPointProperty[6].trim());
                                                    //设置 CategoryId
                if (trackPointProperty.length >= 8) {
                    trackPoint.setTrackPoints(trackPointProperty[7].trim());
                                                    //设置 TrackPoints
                }
                if (trackPointProperty.length >= 9) {
                    trackPoint.setTime(trackPointProperty[8].trim());
                                                    //设置 Time
                }
                //添加 TrackPoint 对象到 TrackPoint 集合
                trackPointList.add(trackPoint);
            }
        }
        return trackPointList;                     //返回 TrackPoint 集合
    }

    /**
     * 获取 Beetle 对象集合
     *
     * @return
     */
    public List<Beetle> getAllBeetleList(Context context) {
        String path = "Beetle.txt";                //获取路径
        //调用文件读取方法，获取字符串
        String content = readFileByByte(context, path);
        if (content != null && !"".equals(content)) {
            //调用填充 Beetle 集合方法，返回 Beetle 集合
            return setBeetleList(content);
        } else {
            return new ArrayList<Beetle>();        //返回默认 Beetle 集合
        }
    }

    /**
     * 获取 Beetle 对象集合
     *
     * @return
     */
    public List<Beetle> getAllBeetleList(Context context, String bigRouteId) {
        String path = bigRouteId + "/Beetle.txt";  //获取路径
        //调用文件读取方法，获取字符串
        String content = readFileByByte(context, path);
        if (content != null && !"".equals(content)) {
            //调用填充 Beetle 集合方法，返回 Beetle 集合
            return setBeetleList(content);
        } else {
            return new ArrayList<Beetle>();        //返回默认 Beetle 集合
        }
    }

    /**
     * 根据 routeId 获取 Beetle 集合
     *
```

```
     * @param context
     * @param routeId
     * @return
     */
   public List<Beetle> getRouteBeetleList(Context context, String routeId)
{
        String path = "SmallRoute/" + routeId + "/route beetle.txt";
                                                     //获取路径
        //调用文件读取方法，获取字符串
        String content = readFileByByte(context, path);
        if (content != null && !"".equals(content)) {
            //调用填充 Beetle 集合方法，返回 Beetle 集合
            return setBeetleList(content);
        } else {
            return new ArrayList<Beetle>();          //返回默认 Beetle 集合
        }
    }

    /**
     * 根据 routeId，bigRouteId 获取 Beetle 集合
     *
     * @param context
     * @param routeId
     * @return
     */
    public List<Beetle> getRouteBeetleList(Context context, String routeId,
            String bigRouteId) {
        String path = bigRouteId + "/SmallRoute/" + routeId
                + "/route_beetle.txt";                //获取路径
        //调用文件读取方法，获取字符串
        String content = readFileByByte(context, path);
        if (content != null && !"".equals(content)) {
            //调用填充 Beetle 集合方法，返回 Beetle 集合
            return setBeetleList(content);
        } else {
            return new ArrayList<Beetle>();          //返回默认 Beetle 集合
        }
    }

    /**
     * 填充 Beetle 集合
     *
     * @param content
     * @return
     */
    public List<Beetle> setBeetleList(String content) {
        ……//该处省略了具体实现代码，将在之后进行具体介绍
    }

    /**
     * 省市关键字查询
     *
     * @param context
     * @param city
     * @param prov
     * @return
     */
    public List<Beetle> searchBeetlsByKeyword(Context context, String city,
```

```
            String prov) {
        ……//该处省略了具体实现代码，将在之后进行具体介绍
    }

    /**
     * 排序
     *
     * @param beetleList
     * @return
     */
    public List<Beetle> sortBeetls(List<Beetle> beetleList) {
        ……//该处省略了具体实现代码，将在之后进行具体介绍
    }

    /**
     * 排序，按大路书名
     *
     * @param context
     * @return
     */
    public List<Route> sortBigRoute(Context context) {
        ……//该处省略了具体实现代码，将在之后进行具体介绍
    }
}
```

下面介绍兴趣点集合去重复的方法，代码如下：

```
public List<PoiPoint> removeRepeatObject(List<PoiPoint> pointList) {
        List<PoiPoint> newPointList = new ArrayList<PoiPoint>();
                                            //初始化 PoiPoint 集合
        Set<String> pointSet = new HashSet<String>();  //初始化 Set 集合
        for (int i = 0; i < pointList.size(); i++) {
            String source = pointList.get(i).getName();
                                            //获取 PoiPoint 的 Name
            if (pointSet.add(source)) {         //判断是否加入 set 集合成功
                newPointList.add(pointList.get(i));
                                        //添加 PoiPoint 对象到 PoiPoint 集合
            }
        }
        return newPointList;            //返回 PoiPoint 集合
    }
```

初始化一个新的 PoiPoint 集合 newPointList，初始化一个 Set 集合，我们知道 Set 集合中存储的值是唯一的，那我们就利用这一点，首先遍历传过来的 pointList 集合，这里我们是根据兴趣点名称来去掉重复的，也就是说名称相同的兴趣点是不允许存在的。source 就是我们获取的兴趣点的名称，用 Set 的 add()方法，把 source 添加到 Set 集合，如果集合中已存在该兴趣点名称，add()方法返回 false，并且不会添加到 Set 集合中，那我们就判断，如果能添加到 Set 集合中的兴趣点名称相对应的兴趣点对象就添加到 newPointList 集合中。这样就达到了去掉重复兴趣点的目的。

下面介绍根据 Route 的 id 查询该 Route 的所有 TrackPoint，代码如下：

```
public List<TrackPoint> getTrackPointListByRouteId(Context context,
            String routeId) {
        String path = "SmallRoute/" + routeId + "/" + routeId
                + "_trackPoint.txt";            //获取文件路径
```

```
        String content = readFileByByte(context, path);
                                            //调用文件读取方法，获取字符串
        if (content != null && !"".equals(content)) {
            //调用填充 TrackPoint 集合，并返回 TrackPoint 集合
            return setTrackPointList(content);
        } else {
            return new ArrayList<TrackPoint>();//返回默认 TrackPoint 集合
        }
    }
```

path 获取文件路径，这里根据传过来的 routeId，拼接出存放 TrackPoint 数据的数据文件，调用 readFileByByte()方法传入 path 获取数据字符串，调用填充 TrackPoint 集合，并返回 TrackPoint 集合。如果返回的字符串为空字符串，或者 content 为 NULL，就返回一个 size 为 0 的 TrackPoint 集合。

下面介绍从数据文件获取的字符串数据转换为 Beetle 对象，代码如下：

```
public List<Beetle> setBeetleList(String content) {
        String[] beetles = content.split("@#@");   //分割字符串
        List<Beetle> beetlesList = new ArrayList<Beetle>();
                                                //初始化 Beetle 集合
        if (beetles.length > 0) {
            for (int i = 0; i < beetles.length; i++) {
                String[] beetleProperty = beetles[i].split("!#!");
                                                //分割字符串
                Beetle beetle = new Beetle();       //初始化 Beetle 对象
                beetle.setId(beetleProperty[0].trim());    //设置 Id
                beetle.setName(beetleProperty[1].trim());  //设置 Name
                beetle.setLat(beetleProperty[2].trim());   //设置 Lat
                beetle.setLon(beetleProperty[3].trim());   //设置 Lon
                beetle.setContacts(beetleProperty[4].trim());//设置 Contacts
                if (beetleProperty.length >= 6) {
                    beetle.setTel(beetleProperty[5].trim());//设置 Tel
                }
                if (beetleProperty.length >= 7) {
                    beetle.setAddress(beetleProperty[6].trim());
                                                        //设置 Address
                }
                if (beetleProperty.length >= 8) {
                    beetle.setZip(beetleProperty[7].trim());//设置 Zip
                }
                if (beetleProperty.length >= 9) {
                    beetle.setFax(beetleProperty[8].trim());//设置 Fax
                }
                if (beetleProperty.length >= 10) {
                    beetle.setEmail(beetleProperty[9].trim());//设置 Email
                }
                //添加 Beetle 对象到 Beetle 集合
                beetlesList.add(beetle);
            }
        }
        return beetlesList;                             //返回 Beetle 集合
    }
```

split()方法根据给定正则表达式的匹配拆分此字符串，这里我们用该方法来获取我们具体想用的数据，然后把获得的字符串数据通过 Beetle 对象的属性的 set()方法赋值给 Beetle

对象的属性，这样就实现了字符串转换 Beetle 对象。

下面介绍根据省市关键字，查询服务区方法，代码如下：

```
public List<Beetle> searchBeetlsByKeyword(Context context, String city,
            String prov) {
        List<Beetle> beetleList = getAllBeetleList(context);
                                                    //获取 Beetle 对象集合
        List<Beetle> beetles = new ArrayList<Beetle>();
                                                    //初始化 Beetle 对象集合
        if (beetleList != null && beetleList.size() > 0) {
            if ("".equals(city) && "".equals(prov)) {
                //调用排序方法，对 Beetle 集合排序
                beetleList = sortBeetls(beetleList);
                return beetleList;                      //返回 Beetle 集合
            }
            for (int i = 0; i < beetleList.size(); i++) {
                String address = beetleList.get(i).getAddress().trim();
                                                        //获取地址
                if ((!"".equals(city) && (address.indexOf(city) >= 0))
                        ||(!"".equals(prov) && (address.indexOf(prov) >= 0))){
                    //添加 Beetle 对象到 Beetle 集合
                    beetles.add(beetleList.get(i));
                }
            }
        }
        //调用排序方法，对 Beetle 集合排序
        beetles = sortBeetls(beetles);
        return beetles;                                 //返回 Beetle 集合
    }
```

因为我们不是在数据库中操作这些数据，而是通过文件获取的数据，那我们只能在内存中来操作数据，以获取我们想要的数据。beetleList 获取了所有 Beetle 对象集合，在初始化一个新的 Beetle 对象集合 beetles，我们遍历 beetleList 集合，判断集合中的每个对象中、省市及地址，是否包含我们的关键字，如果包含，就把该对象添加到 beetles 集合中，这就是一个按照省市关键字查询。

下面我们介绍根据 Beetle（服务区实体类）的地址进行 Beetle 集合排序，代码如下：

```
public List<Beetle> sortBeetls(List<Beetle> beetleList) {
        List<Beetle> newBeetles = new ArrayList<Beetle>();
                                                //初始化 Beetle 对象集合
        String[] address = new String[beetleList.size()];  //初始化地址数组
        for (int i = 0; i < beetleList.size(); i++) {
            address[i] = beetleList.get(i).getAddress().trim();
                                                //添加地址到集合
        }
        Comparator cmp = Collator.getInstance(java.util.Locale.CHINA);
                                                //初始化 Comparator
        Arrays.sort(address, cmp);              //数组排序方法
        for (int i = 0; i < address.length; i++) {
            for (int j = 0; j < beetleList.size(); j++) {
                if (address[i].equals(beetleList.get(j).getAddress().trim())) {
                    newBeetles.add(beetleList.get(j));
                                            //添加 Beetle 对象到 Beetle 集合

                }
```

```
            }
        }
        return newBeetles;                    //返回 Beetle 集合
    }
```

首先要取出 Beetle 集合中的地址，因为我们要根据地址排序，调用 Collator.getInstance()方法获取所需语言环境的 Collator；Locale 对象表示了特定的地理、政治和文化地区；Locale.CHINA 表示为中国创建了一个 Locale 对象；Arrays.sort()就是排序的核心方法，传入排序的数组和定义排序的规则，通过 Arrays.sort()对该数组进行排序，然后遍历排序后的地址字符串数组，对应 beetleList 中的对象的地址，将对应好的添加到新集合中，排序完成。

下面我们介绍根据 Route 路书名称进行 Route 集合排序，代码如下：

```
public List<Route> sortBigRoute(Context context) {
        List<Route> bigRouteList = getBigRouteList(context);
                                              //获取 Route 对象集合
        List<Route> newBigRoutes = new ArrayList<Route>();
                                              //初始化 Route 对象集合
        String[] bigRouteNames = new String[bigRouteList.size()];
                                              //初始化名称数组
        for (int i = 0; i < bigRouteList.size(); i++) {
            bigRouteNames[i] = bigRouteList.get(i).getName().trim();
                                              //添加名称到数组
        }
        Comparator cmp = Collator.getInstance(java.util.Locale.CHINA);
                                              //初始化 Comparator
        Arrays.sort(bigRouteNames, cmp);    //数组排序方法
        for (int i = 0; i < bigRouteNames.length; i++) {
            for (int j = 0; j < bigRouteList.size(); j++) {
                if (bigRouteNames[i].equals(bigRouteList.get(j).getName()
                        .trim())) {
                    newBigRoutes.add(bigRouteList.get(j));
                                              //添加 Route 对象到 Route 集合
                }
            }
        }

        return newBigRoutes;                  //返回 Route 集合
    }
```

这个排序的原理和 Beetle 集合排序原理一样，首先要取出 Route 集合中 Route 的名称，把名称添加到字符串数组 bigRouteNames 中，然后调用 Arrays.sort()方法进行排序，排序完成后，又根据对应的名称找到该 Route 对象，将 Route 对象添加到新集合中去，return 返回排序好的新集合。

13.8　公共类的代码实现

在服务区窗体可以根据省份信息搜索数据，省份信息保存在公有数据类中，具体代码如下。

```
package com.tyhj;
```

```
public class StaticString {
    public static String pro[]={"安徽","北京","重庆","福建","甘肃","广东",
    "广西","贵州","海南","河北","黑龙江",
       "河南","香港","湖北","湖南","江苏","江西","吉林","辽宁","澳门","内蒙古",
        "宁夏","青海","山东","上海","山西",
        "陕西","四川","台湾","天津","新疆","西藏","云南","浙江"};
    public static String nick="";
}
```

13.9　欢迎窗体类的设计及实现

应用启动后是一个欢迎窗体，欢迎窗体全屏显示，2 秒钟后自动出现 Logo 窗体。欢迎窗体的实现流程如下：

（1）在 onCreate()方法中初始化窗体信息。例如，隐藏标题栏、状态栏，加载布局文件。

（2）自定义 Thread 类实现 2 秒钟后跳转。

13.9.1　欢迎窗体的框架设计

首先介绍欢迎窗体的代码设计框架，有利于大家很好地了解代码流程，具体代码如下：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
/**
 * 第一个欢迎窗体
 **/

public class Welcome extends Activity{

    /**
     * 重写 Activity 中的 onCreate 的方法
     * 该方法是在 Activity 创建时被系统调用，是一个 Activity 生命周期的开始
     * @param savedInstanceState: 保存 Activity 的状态的
     *      Bundle 类型的数据与 Map 类型的数据相似，都是以 key-value 的形式存储数据的
     * @return
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        ……//此处省略了初始化窗体事件，将在之后进行介绍
    }
    /**
     * 欢迎界面，2 秒钟后切换
     * @param
     * @return
     */
    public void welcome() {
        new Thread(new Runnable() {                //创建线程
            public void run() {           //实现 Runnable 的 run()方法，即线程体
                try {
```

```
                    Thread.sleep(2000);             //欢迎界面暂停 2 秒钟
                    Message m = new Message();      //创建 Message 对象
                    logHandler.sendMessage(m);      //将消息放到消息队列中
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }).start();//启动线程
    }

    //执行接收到的消息，执行的顺序是按照队列进行，即先进先出
    Handler logHandler = new Handler() {
        public void handleMessage(Message msg) {
            welcome1();                                     //显示 Logo 界面
        }
    };
    /**
     * 显示 Logo 界面
     * @param
     * @return
     */
    public void welcome1() {
        Intent it=new Intent();//实例化 Intent
        it.setClass(Welcome.this, Welcome1.class);      //设置 Class
    startActivity(it);//启动 Activity
    Welcome.this.finish();//结束 Welcome Activity
    }

    /**
     * 键盘按键按下时触发该方法
     * @param keyCode：被按下的键值即键盘码
     *        event：按键事件的对象
     * @return
     */
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if(keyCode==4 ){                                //按下“返回”按键
            android.os.Process.killProcess(android.os.Process.myPid());
                                                        //让程序完全退出应用
        }
        return super.onKeyDown(keyCode, event);
    }
}
```

13.9.2　欢迎窗体的初始化工作

欢迎窗体是以全屏的方式展现，需要在 Activity 的 onCreate()方法中隐藏标题栏和状态栏。窗体初始化代码如下：

```
protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        //返回当前 Activity 的 Window 对象,Window 类中概括了 Android 窗口的基本属性
          和基本功能
        final Window win = getWindow();
```

```
        //隐藏状态栏
    win.setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,WindowManager.
    LayoutParams.FLAG_FULLSCREEN);
        requestWindowFeature(Window.FEATURE_NO_TITLE); //隐藏标题栏
        this.setContentView(R.layout.welcome);          //设置布局资源
        //获取 welcome.xml 中 id 为 wpic 的 ImageView 组件
        ImageView iv=(ImageView)this.findViewById(R.id.wpic);
        iv.setImageResource(R.drawable.welcome);//设置 ImageView 上显示的资源
        welcome();                                    //欢迎界面
    }
```

13.10　Logo 窗体类的设计及实现

Logo 窗体展示应用 Logo 信息，2 秒钟后自动显示精品线路列表信息窗体。

13.10.1　Logo 窗体的框架设计

Logo 窗体的代码流程如下：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
/**
 * 第二个 Logo 窗体
 * */
public class Welcome1 extends Activity{
    /**
     * 重写 Activity 中的 onCreate 的方法
     * 该方法是在 Activity 创建时被系统调用，是一个 Activity 生命周期的开始
     * @param savedInstanceState：保存 Activity 的状态的
     *      Bundle 类型的数据与 Map 类型的数据相似，都是以 key-value 的形式存储数据的
     * @return
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        //返回当前 Activity 的 Window 对象，Window 类中概括了 Android 窗口的基本属性
          和基本功能
        final Window win = getWindow();
        //隐藏状态栏
    win.setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,WindowManager.
    LayoutParams.FLAG_FULLSCREEN);
        requestWindowFeature(Window.FEATURE_NO_TITLE); //隐藏标题栏
        this.setContentView(R.layout.welcome1);         //设置布局资源
        //获取 welcome.xml 中 id 为 wpic 的 ImageView 组件
        ImageView iv=(ImageView)this.findViewById(R.id.wpic);
        iv.setImageResource(R.drawable.welcome1);//设置 ImageView 上显示的资源
        tripListView();                                //显示线路列表窗体
    }

    /**
```

```
     * 欢迎界面,2 秒钟后切换
     * @param
     * @return
     */
    public void tripListView() {
        new Thread(new Runnable() {              //创建线程
            public void run() {          //实现 Runnable 的 run()方法，即线程体
                try {
                    Thread.sleep(2000);  //欢迎界面暂停 2 秒钟
                    Message m = new Message(); //创建 Message 对象
                    logHandler.sendMessage(m); //将消息放到消息队列中
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }).start();//启动线程
    }

    //执行接收到的消息，执行的顺序是按照队列进行，即先进先出
    Handler logHandler = new Handler() {
        public void handleMessage(Message msg) {
            tripList();                                 //显示线路列表界面
        }
    };
    /**
     * 线路列表界面
     * @param
     * @return
     */
    public void tripList() {
        Intent it=new Intent();//实例化 Intent
        it.setClass(Welcome1.this, TripList.class);    //设置 Class
    startActivity(it);//启动 Activity
    Welcome1.this.finish();//结束 Welcome1 Activity
    }
    /**
     * 键盘按键按下是触发该方法
     * @param keyCode: 被按下的键值即键盘码
     *       event: 按键事件的对象
     * @return
     */
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        ……//此处省略了 onKeyDown()方法的代码实现，将在之后进行介绍
    }
}
```

13.10.2　onKeyDown 事件处理

onKeyDown 事件是键盘按键按下事件，无论是欢迎窗体还是 Logo 窗体，当按下键盘上的“返回”按钮时，退出当前的应用。“返回”按键的 keyCode 码是 4，通过 killProcess()方法结束当前应用进程。具体代码如下：

```
public boolean onKeyDown(int keyCode, KeyEvent event) {
        if(keyCode==4 ){                        //按下“返回”按键
```

```
            android.os.Process.killProcess(android.os.Process.myPid());
                                                    //让程序完全退出应用
        }
        return super.onKeyDown(keyCode, event);
    }
```

13.11　精品线路列表窗体类的设计及实现

精品线路列表窗体展示路书列表信息。单击列表中的某一项，进入线路详情展示窗体。本窗体的实现涉及以下几个技术点：

- 动态创建用来显示商品列表的组件 ListView。
- 通过 ListView 的 setOnItemClickListener 事件监听单击列表项事件。
- 布局文件分为两部分，一部分是精品线路列表展示的布局文件 res/layout/triplist.xml，另一部分是精品线路列表项布局文件 res/layout/trippoilistrow.xml。当单击选择某一个列表项或者列表项获取某一个焦点时，列表项会改变背景，此效果通过 res/layout/trippoilistviewbg.xml 实现。

13.11.1　精品线路列表窗体的框架设计

精品线路列表窗体的设计思路及实现流程如下：

（1）首先通过 WAnalysisFile 类中提供的 sortBigRoute()方法，获取精品线路信息列表，并将其保存到商品集合 tripLists 变量中。

（2）然后创建 SimpleAdapter 适配器，数据源是 tripLists 集合中的数据信息。

（3）最后创建 ListView 组件，为该组件添加 SimpleAdapter 适配器，以便显示数据。

这里我们看一下精品线路列表的详细代码实现：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
/**
 * 线路列表窗体
 * */
public class TripList extends Activity {
    private LinearLayout tripListLayout;
            //声明 LinearLayout 类型变量，用来显示路书列表的 LinearLayout 布局
    private ListView myListView;
            //声明 ListView 变量，用来显示路书列表的 ListView 组件
    private TextView tripName;          //声明 TextView 变量，用来显示路书名称
    private List<Route> tripLists;      //声明 List 变量，用来保存路书列表

    /**
     * 重写 Activity 中的 onCreate 的方法。该方法是在 Activity 创建时被系统调用，是
       一个 Activity 生命周期的开始  *
     * @param savedInstanceState
     *           ：保存 Activity 的状态的。 Bundle 类型的数据与 Map 类型的数据相似，
                  都是以 key-value 的形式存储数据的
     * @return
```

```
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setTitle(R.string.title);
                                    //设置窗体标题，为 String.xml 中 title 的值
        this.setContentView(R.layout.triplist);    //设置布局资源
        setTripPoiList();                          //显示线路列表方法
    }

    /**
     * 显示线路列表
     *
     * @param
     * @return
     */
    public void setTripPoiList() {
        //从 triplist.xml 中获取名字为 tripName 的 TextView 对象
        tripName = (TextView) this.findViewById(R.id.tripName);
        tripName.setText("精品旅游线路推荐");     //设置 tripName 显示文字
        //从 triplist.xml 中获取名字为 tripListLayout 的 LinearLayout 对象
        tripListLayout = (LinearLayout) this.findViewById(R.id.tripListLayout);
        myListView = new ListView(this);       //动态创建 ListView 对象
        //设置布局参数
        LinearLayout.LayoutParams param3 = new LinearLayout.LayoutParams(

                LinearLayout.LayoutParams.FILL_PARENT,
                LinearLayout.LayoutParams.FILL_PARENT);
        myListView.setCacheColorHint(Color.WHITE);
                                        //列表拖曳过程中高亮颜色，默认是黑色
       //将 myListView 添加到 tripListLayout 上，按照 param3 方式布局
        tripListLayout.addView(myListView, param3);
        //创建适配器
        SimpleAdapter adapter = new SimpleAdapter(this, getData(),
                R.layout.trippoilistrow,
                new String[] { "title", "tel", "img", }, new int[] {
                        R.id.poiTitle, R.id.poiTel, R.id.poiImg });
        myListView.setAdapter(adapter);    //为 myListView 添加适配器
        //setViewBinder()方法设置 binder 用于绑定数据到视图，参数为用于绑定数据到
          视图的 binder
        adapter.setViewBinder(new ViewBinder() {
            @Override
            public boolean setViewValue(View arg0, Object arg1,
                    String textRepresentation) {
                //TODO Auto-generated method stub
                if ((arg0 instanceof ImageView) & (arg1 instanceof Bitmap)) {
                    ImageView imageView = (ImageView) arg0;
                    Bitmap bitmap = (Bitmap) arg1;
                    imageView.setImageBitmap(bitmap);
                                              //设置 imageView 组件显示的图片
                    return true;
                } else {
                    return false;
                }
            }
        });
```

```
        //ListView 列表项单击事件
        myListView.setOnItemClickListener(new OnItemClickListener() {
            //position 指在 ListView 里的位置，id 是 view 的资源 id
            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1,
                    int position, long id) {
                //TODO Auto-generated method stub
                Intent it = new Intent();           //实例化 Intent
                Bundle poiMsg = new Bundle();       //实例化 Bundle
                it.setClass(TripList.this, TripDetail.class);//设置 Class
                //将当前 ListView 被单击的项目的 Route 对象放到 Bundle 中
                poiMsg.putSerializable("tripObj", (Serializable) tripLists
                        .get(position));
                it.putExtras(poiMsg);
                            //将该对象作为参数传递给下一个窗体，即 TripDetail
                startActivity(it);                  //启动 Activity
            }
        });
    }

    /**
     * 获取线路信息
     *
     * @param
     * @return
     */
    private List<Map<String, Object>> getData() {
        ……//此处省略了方法功能实现，将在之后进行介绍
        }
        return list;
    }
}
```

13.11.2　精品线路列表的 ListView 数据填充

精品线路列表展示在 ListView 组件上，精品线路列表数据保存在 tripLists 集合中，通过 getData()方法，将线路名称、公里数、线路所在省以(key, value)的形式保存到集合中，作为 SimpleAdapter 的数据源。具体实现代码如下：

```
private List<Map<String, Object>> getData() {
        List<Map<String, Object>> list = new ArrayList<Map<String,
Object>>();//创建 List 对象

        WAnalysisFile readFile = new WAnalysisFile();
                                    //创建 WAnalysisFile 自定义类对象
        tripLists = readFile.sortBigRoute(TripList.this);
                                    //排序，按大路书名称

        for (int i = 0; i < tripLists.size(); i += 1){ //循环获取路书信息
            Map<String, Object> map = new HashMap<String, Object>();
            Route trip = tripLists.get(i);          //获取集合中的 Route 对象
            map.put("title", trip.getName() + ", " + trip.getMileage() + "
            公里");
```

```
            map.put("tel", trip.getCity());

            String tripImg = trip.getImage();
            InputStream iso = null;
            try {
                iso = this.getAssets().open(trip.getId() + "/" + tripImg);
                                                    //获取 assets 下图片
            } catch (IOException e) {
                //TODO Auto-generated catch block
                e.printStackTrace();
            }
            Bitmap bitmap = null;
            bitmap = BitmapFactory.decodeStream(iso);
            map.put("img", bitmap);

            list.add(map);
        }
        return list;
    }
```

精品线路列表窗体运行效果如图 13.19 所示。



13.19　精品线路列表窗体

13.12　精品线路详情窗体类的设计及实现

精品线路详情窗体展示线路详细信息，窗体上有 3 个按钮，分别为“详情”、“兴趣点”、“服务区”。单击“详情”按钮进入分段路数详情列表展示；单击“兴趣点”按钮进入兴趣点列表展示；单击“服务区”按钮进入服务区列表展示。

13.12.1　精品线路详情窗体的框架设计

该窗体的代码实现流程及注意事项如下：

- 获取传递过来参数的 Bundle 对象，通过 Bundle 的 getSerializable()方法获取路书对象 Route，通过 Route 类中提供的 getXXX()方法获取属性信息，显示在窗体组件上。
- 路书图片名称通过 route.getImage()方法获得，因为这里路书图片需要在列表显示，还需要在详情页显示，所以准备了不同尺寸的图片，通过图片名称来区分。详情页图片为大图，图片名字为 XXX_b.jpg，图片展示窗体显示的是原图，原图名字为 y_XXX.jpg，精品线路列表中的图片为小图，route.getImage()方法默认获取的图片名即为小图的名称。所以这里显示图片名需要处理一下，如 route.getImage().split("[.]")[0] + "_b.jpg"。
- 路书指数分为 11 级，0～10。这 11 级指数对应 11 张图片，数据库中存储的指数为整数类型，需要通过该类中自定义方法 getPicDrawId()进行转换。

精品线路详情窗体 Java 代码如下：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
/**
 * 线路详情窗体
 **/
public class TripDetail extends Activity {
    private TextView tripName;      //声明 TextView 变量，用来显示路书名称
    private TextView startPos;      //声明 TextView 变量，用来显示起点
    private TextView endPos;        //声明 TextView 变量，用来显示终点
    private TextView tripPro;       //声明 TextView 变量，用来显示省份
    private TextView distance;      //声明 TextView 变量，用来显示里程
    private TextView suggest;       //声明 TextView 变量，用来显示建议行程
    private TextView bestSeason;    //声明 TextView 变量，用来显示最佳季节
    private ImageView tripPic;      //声明 ImageView 变量，用来显示路书图片
    private ImageView jsnd;         //声明 ImageView 变量，用来显示路况指数
    private ImageView fgzs;         //声明 ImageView 变量，用来显示风光指数
    private ImageView rwzs;         //声明 ImageView 变量，用来显示人文指数
    private ImageView mszs;         //声明 ImageView 变量，用来显示美食指数
    private TextView tripDesc;      //声明 TextView 变量，用来显示路书描述
    private TextView tripTip;       //声明 TextView 变量，用来显示路书提示
    private ImageButton xqButt;     //声明 ImageButton 变量，详情按钮
    private ImageButton xqdButt;    //声明 ImageButton 变量，兴趣点按钮
    private ImageButton sdButt;     //声明 ImageButton 变量，服务站按钮
    private ImageView picImgView;   //声明 ImageView 变量，路书图片
    private TextView tgzs;          //声明 TextView 变量，指数文字
    private TextView js;            //声明 TextView 变量，路书介绍文字
    private TextView ts;            //声明 TextView 变量，路书提示文字

    /**
     * 重写 Activity 中的 onCreate 的方法。该方法是在 Activity 创建时被系统调用，是
       一个 Activity 生命周期的开始
     * @param savedInstanceState
     *          ：保存 Activity 的状态的。 Bundle 类型的数据与 Map 类型的数据相似，
                都是以 key-value 的形式存储数据的
     * @return
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
```

```
        this.setTitle(R.string.title);      //设置窗体标题栏文字
        this.setContentView(R.layout.tripdetail);
                                            //加载 tripdetail.xml 布局资源
        setTripDetail();
    }

    /**
     * 获取路书详细信息
     *
     * @param
     * @return
     */
    public void setTripDetail() {
        Bundle tripObj = getIntent().getExtras();
                                            //获取前一个窗体传递过来的参数
        final Route route = (Route) tripObj.getSerializable("tripObj");
                                            //将参数转化成序列化对象
        tripName = (TextView) this.findViewById(R.id.tripName);
                                            //获取路书名称的 TextView 对象
        tripName.setText(route.getName());  //设置路书名称TextView组件显示文字
        startPos = (TextView) this.findViewById(R.id.startPos);
                                            //获取起点的 TextView 对象
        startPos.setText("起      点: " + route.getStartPointName());
                                            //设置起点 TextView 组件显示文字
        endPos = (TextView) this.findViewById(R.id.endPos);
                                            //获取终点的 TextView 对象
        endPos.setText("终      点: " + route.getEndPointName());
                                            //设置终点 TextView 组件显示文字
        tripPro = (TextView) this.findViewById(R.id.tripPro);
                                            //获取所在省的 TextView 对象
        tripPro.setText("所  在  省: " + route.getCity());
                                            //设置所在省 TextView 组件显示文字
        distance = (TextView) this.findViewById(R.id.distance);
                                            //获取里程的 TextView 对象
        distance.setText("里      程: " + route.getMileage() + "公里");
                                            //设置里程 TextView 组件显示文字
        suggest = (TextView) this.findViewById(R.id.suggest);
                                            //获取建议行程的 TextView 对象
        //设置建议行程 TextView 组件显示文字
        suggest.setText("建议行程: " + route.getProposedItinerary() + " 天");
        bestSeason = (TextView) this.findViewById(R.id.bestSeason);
                                            //获取最佳季节的 TextView 对象
        bestSeason.setText("最佳季节: " + route.getBestSeason());
                                        //设置最佳季节 TextView 组件显示文字
        tripPic = (ImageView) this.findViewById(R.id.tripPic);
                                            //获取路书图片的 ImageView 对象
        String tripImg = route.getImage().split("[.]")[0] + "_b.jpg";
                                            //获取图片称
        InputStream iso;
        try {
            iso = this.getAssets().open(route.getId() + "/" + tripImg);
                                            //打开 assets 下的图片
            Bitmap bitmap = null;
            bitmap = BitmapFactory.decodeStream(iso);
            tripPic.setImageBitmap(bitmap);//设置 tripPic 上显示的图片
        } catch (IOException e1) {
            //TODO Auto-generated catch block
```

```
        e1.printStackTrace();
    }
    tgzs = (TextView) this.findViewById(R.id.tgzs);
                                   //获取线路指数 TextView 组件
    TextPaint tgzstp = tgzs.getPaint();
    tgzstp.setFakeBoldText(true);      //设置线路指数 TextView 为粗体显示
    js = (TextView) this.findViewById(R.id.js);
                                   //获取线路介绍 TextView 组件
    TextPaint jstp = js.getPaint();
    jstp.setFakeBoldText(true);        //设置线路介绍 TextView 为粗体显示
    ts = (TextView) this.findViewById(R.id.ts);
                                   //获取线路提示 TextView 组件
    TextPaint tstp = ts.getPaint();
    tstp.setFakeBoldText(true);        //设置线路提示 TextView 为粗体显示

    //4 个指数
    jsnd = (ImageView) this.findViewById(R.id.jsnd);
                                   //获取路况指数 ImageView 组件
    //设置路况指数显示的图片，getPicDrawId()为自定义方法，实现将指数数字转化为
      对应的图片
    jsnd.setBackgroundResource(getPicDrawId(route.getRecommend()));
    fgzs = (ImageView) this.findViewById(R.id.fgzs);
                                   //获取风光指数 ImageView 组件
    fgzs.setBackgroundResource(getPicDrawId(route.getScenic()));
                                   //设置风光指数显示的图片
    rwzs = (ImageView) this.findViewById(R.id.rwzs);
                                   //获取人文指数 ImageView 组件
    rwzs.setBackgroundResource(getPicDrawId(route.getRenwen()));
                                   //设置人文指数显示的图片
    mszs = (ImageView) this.findViewById(R.id.mszs);
                                   //获取美食指数 ImageView 组件
    mszs.setBackgroundResource(getPicDrawId(route.getFood()));
                                   //设置美食指数显示的图片
    tripDesc = (TextView) this.findViewById(R.id.tripDesc);
                                   //获取介绍 TextView 组件
    //介绍文字中##为分段标志，所以替换成换行符
    String tripDescStr = route.getDesc().replace("##", "\n\n");
    tripDescStr = tripDescStr.replace("\"", "”");
                                   //将半角字符替换成全角字符显示
    tripDescStr = tripDescStr.replace(",", "，");
    tripDescStr = tripDescStr.replace("(", "（");
    tripDescStr = tripDescStr.replace(")", "）");
    tripDescStr = tripDescStr.replace(";", "；");
    tripDesc.setText(tripDescStr);     //设置介绍 TextView 组件显示的文字
    tripTip = (TextView) this.findViewById(R.id.tripTip);
                                   //获取提示 TextView 组件
    String tripTipStr = route.getTip().replace("##", "\n\n");
                           //提示文字中##为分段标志，所以替换成换行符
    tripTipStr = tripTipStr.replace("\"", "”");
                                   //将半角字符替换成全角字符显示
    tripTipStr = tripTipStr.replace(",", "，");
    tripTipStr = tripTipStr.replace("(", "（");
    tripTipStr = tripTipStr.replace(")", "）");
    tripTipStr = tripTipStr.replace(";", "；");
    tripTip.setText(tripTipStr + "\n");//设置提示 TextView 组件显示的文字
```

```
        xqButt = (ImageButton) this.findViewById(R.id.xqButt);
                                    //获取详情 ImageButton 组件
        xqButt.setOnClickListener(new OnClickListener() {
                                    //详情 ImageButton 组件的单击事件
                @Override
                public void onClick(View v) {
                    //TODO Auto-generated method stub
                    Intent it = new Intent();       //实例化 Intent
                    Bundle tripMsg = new Bundle(); //实例化 Bundle
                    it.setClass(TripDetail.this, TripSegment.class);
                                                    //设置 Class
                    tripMsg.putString("tripId", route.getId());
                                            //将路书 id 保存到 Bundle 中
                    tripMsg.putString("tripName",  route.getName());
                                            //将路书名字保存到 Bundle 中
                    it.putExtras(tripMsg);
                                //将 tripMsg 对象作为参数传递给下一个窗体
                    startActivity(it);      //启动 Activity
                }

            });
        xqdButt = (ImageButton) this.findViewById(R.id.xqdButt);
                                //获取兴趣点 ImageButton 组件
        xqdButt.setOnClickListener(new OnClickListener() {
                                //兴趣点 ImageButton 组件的单击事件

                @Override
                public void onClick(View v) {
                    //TODO Auto-generated method stub
                    Intent it = new Intent();       //实例化 Intent
                    Bundle tripMsg = new Bundle(); //实例化 Bundle
                    it.setClass(TripDetail.this, TripPoiList.class);
                                                    //设置 Class
                    tripMsg.putString("tripId", route.getId());
                                            //将路书 id 保存到 Bundle 中
                    tripMsg.putString("tripName", route.getName());
                                            //将路书名字保存到 Bundle 中
                    it.putExtras(tripMsg);
                                //将 tripMsg 对象作为参数传递给下一个窗体
                    startActivity(it);      //启动 Activity
                }

            });
        sdButt = (ImageButton) this.findViewById(R.id.sdButt);
                                //获取服务区 ImageButton 组件
        sdButt.setOnClickListener(new OnClickListener() {
                                //服务区 ImageButton 组件的单击事件

                @Override
                public void onClick(View v) {
                    //TODO Auto-generated method stub
                    Intent it = new Intent();       //实例化 Intent
                    it.setClass(TripDetail.this, DZDealersList.class);
                                                    //设置 Class
                    startActivity(it);              //启动 Activity
                }

            });
        picImgView = (ImageView) this.findViewById(R.id.tripPic);
                                //获取路书图片 ImageView 组件
```

```
        picImgView.setOnClickListener(new OnClickListener() {
                                                //路书图片 ImageView 组件的单击事件

                    @Override
                    public void onClick(View v) {
                                                //单击路书图片，显示路书图片的原尺寸效果
                        //TODO Auto-generated method stub
                        ……//此处省略了方法功能实现，将在之后进行介绍
                    }

                });
    }

    /**
     * 获取指数数字对应的指数图片
     *
     * @param zs
     *          ：指数数字信息
     * @return
     */
    public int getPicDrawId(int zs) {
        int zsint = 0;
        switch (zs) {
        case 0:
            zsint = R.drawable.xing0;
            break;
        case 1:
            zsint = R.drawable.xing1;
            break;
        case 2:
            zsint = R.drawable.xing2;
            break;
        case 3:
            zsint = R.drawable.xing3;
            break;
        case 4:
            zsint = R.drawable.xing4;
            break;
        case 5:
            zsint = R.drawable.xing5;
            break;
        case 6:
            zsint = R.drawable.xing6;
            break;
        case 7:
            zsint = R.drawable.xing7;
            break;
        case 8:
            zsint = R.drawable.xing8;
            break;
        case 9:
            zsint = R.drawable.xing9;
            break;
        case 10:
            zsint = R.drawable.xing10;
            break;
        }
        return zsint;
    }
}
```

13.12.2　展示图片详情窗体功能实现

单击线路详情展示窗体中的图片时，进入图片详情展示窗体，此时需要传递图片路径作为参数，以便在图片详情窗体显示图片信息。具体代码如下：

```
picImgView.setOnClickListener(new OnClickListener() {
                                        //路书图片 ImageView 组件的单击事件

                    @Override
                    public void onClick(View v) {
                                        //单击路书图片，显示路书图片的原尺寸效果
                        //TODO Auto-generated method stub
                        Intent it = new Intent();                    //实例化 Intent
                        Bundle tripDetailPic = new Bundle(); //实例化 Bundle
                        it.setClass(TripDetail.this, TripDetailPic.class);
                                                                    //设置 Class
                        String tripImg = route.getId() + "/y "
                                + route.getImage().split("[.]")[0] + ".jpg";
                                                                    //大图图片的路径
                        tripDetailPic.putString("tripImgPath",  tripImg);
                                //将图片路径保存到 tripDetailPic 中
                        it.putExtras(tripDetailPic);
                                //将 tripDetailPic 对象作为参数传递给下一个窗体
                        startActivity(it);                          //启动 Activity
                    }

                });
```

运行效果如图 13.20 所示。



图 13.20　线路详情展示

13.13　详情图片窗体窗体类的设计及实现

单击详情窗体中的图片，显示原尺寸图片，图片的信息通过 Bundle 传递到本窗体。Java 代码如下：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
/**
 * 单击线路详情中的图片，展示窗体
 * */
public class TripDetailPic extends Activity {
    private ImageView tripPic;//声明 ImageView 变量

    /**
     * 重写 Activity 中的 onCreate 的方法。 该方法是在 Activity 创建时被系统调用，是
     一个 Activity 生命周期的开始
     *
     * @param savedInstanceState
     *            : 保存 Activity 的状态的。 Bundle 类型的数据与 Map 类型的数据相似，
     都是以 key-value 的形式存储数据的
     * @return
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.tripdetailpic);   //设置窗体布局资源

        tripPic = (ImageView) this.findViewById(R.id.tripDetailPic);
                                             //获取图片的 ImageView 组件

        Bundle tripDetailPic = getIntent().getExtras();
                                             //获取传递过来的 Bundle

        String tripDetailName = tripDetailPic.getString("tripName");
                                             //获取路书名参数

        String tripPicPath = tripDetailPic.getString("tripImgPath");
                                             //获取路书图片路径

        InputStream iso;
        try {
            iso = this.getAssets().open(tripPicPath);   //打开 assets 下的图片
            Bitmap bitmap = null;
            bitmap = BitmapFactory.decodeStream(iso);
            tripPic.setImageBitmap(bitmap);
        } catch (IOException e1) {
            //TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }

}
```

运行效果如图 13.21 所示。



图 13.21　线路图片展示

13.14　分段详情展示窗体类的设计及实现

单击精品线路展示窗体的“详情”按钮，进入线路分段详情展示窗体，该窗体展示了每一段线路的公里数和描述信息，单击该窗体上的“地图”按钮，在地图上展示该线路及兴趣点信息。分段详情展示功能的实现，涉及动态布局的实现。

动态布局的实现是指一条线路有可能会有多条分段线路信息，这里分段信息的布局不是通过 XML 定义的，而是通过代码动态实现的。这种方式读者一定要掌握。

13.14.1　分段详情展示窗体的框架设计

分段详情展示窗体代码实现流程如下：

（1）通过 Bundle 获取传递过来的参数获取路书名保存到变量 TripName 中，以及路书 id 保存到变量 tripId 中。

（2）通过 WAnalysisFile 类中的 getRouteList()方法获取指定路书 id 的分段信息，保存到 routList 集合中。

（3）动态的添加分段信息到窗体上。

Java 代码如下：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
/**
 * 线路分段展示窗体
```

```
 * */
public class TripSegment extends Activity {

    private TextView tripSegName;  //声明 TextView 变量，用来显示分段路书名称
    private LinearLayout tripSegsLayout;
                                   //声明 LinearLayout 变量，用来显示分段的布局
    private LinearLayout titleSegLayout;
                         //声明 LinearLayout 变量，用来显示标题和地图按钮的布局
    private TextView titleSegTitle; //声明 TextView 变量，用来显示分段的标题
    private TextView titleSegTitleNum;
                                   //声明 TextView 变量，用来显示分段列表的标号
    private ImageButton mapButt;   //声明 ImageButton 变量，用来显示地图按钮
    private TextView tripDescSeg;  //声明 TextView 变量，用来显示分段描述
    private ImageView tripPicSeg;  //声明 ImageView 变量，用来显示分段图片

    /**
     * 重写 Activity 中的 onCreate 的方法。该方法是在 Activity 创建时被系统调用，是
     一个 Activity 生命周期的开始
     *
     * @param savedInstanceState
     *           : 保存 Activity 的状态的。 Bundle 类型的数据与 Map 类型的数据相似，
     都是以 key-value 的形式存储数据的
     * @return
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setTitle(R.string.title);              //设置标题文字
        this.setContentView(R.layout.tripsegment); //加载布局资源
        getTripSegments();                          //获取路书的分段信息并显示
    }

    /**
     * 获取路书的分段信息并显示
     *
     * @param
     * @return
     */
    public void getTripSegments() {
        Bundle bundle = getIntent().getExtras();           //获取 Bundle
        final String TripName = bundle.getString("tripName");
                                                            //获取路书名参数
        final String tripId = bundle.getString("tripId");    //获取路书 id 参数
        tripSegName = (TextView) this.findViewById(R.id.tripName);
                                      //获取路书分段名称 TextView 组件
        tripSegName.setText(TripName); //将路书分段名称显示在 tripSegName
        //动态加载路书分段信息
        tripSegsLayout = (LinearLayout) this.findViewById(R.id.tripSegs);
                                      //获取路书分段信息的 LinearLayout
        WAnalysisFile readFile = new WAnalysisFile();
                                      //声明 WAnalysisFile 类对象
        List<Route> routList = readFile.getRouteList(this, tripId);
                                      //获取分段信息列表
        for (int i = 0; i < routList.size(); i += 1) {   //循环获取分段信息
            ……//此处省略了动态生成路书分段列表信息代码，将在之后进行介绍
```

```
            }
        }
}
```

13.14.2　动态显示线路分段列表功能的实现

多数情况下，布局的实现都是通过 XML 文件实现，但这里分段列表是动态的，所以无法通过布局文件 XML 实现。需要在代码中创建一个组件，通过 LinearLayout.LayoutParams 设置布局参数，然后通过 addView()方法将组件添加到容器上。动态显示线路分段列表的代码如下：

```
for (int i = 0; i < routList.size(); i += 1) { //循环获取分段信息
            final Route route = routList.get(i);
            //动态生成标题和地图按钮，以及所在的 layout
            titleSegLayout = new LinearLayout(this);
                                    //创建 LinearLayout 对象，用来显示分段信息
            titleSegLayout.setOrientation(LinearLayout.HORIZONTAL);
                                    //设置布局方式为水平布局
            //创建布局参数
            LinearLayout.LayoutParams titleSegLayoutParam = new Linear
            Layout.LayoutParams(
                    LayoutParams.WRAP CONTENT, LayoutParams.WRAP CONTENT);
            titleSegLayoutParam.setMargins(0, 8, 0, 0); //设置边距
            //titleSegLayout 添加 titleSegLayoutParam 布局参数
            titleSegLayout.setLayoutParams(titleSegLayoutParam);
            //标题前的序号
            titleSegTitleNum = new TextView(this);      //创建 TextView 组件
            //创建布局参数
            LinearLayout.LayoutParams titleSegNumParam = new LinearLayout.
            LayoutParams(
                    26, LayoutParams.WRAP CONTENT);
            //titleSegTitleNum 添加 titleSegNumParam 布局参数
            titleSegTitleNum.setLayoutParams(titleSegNumParam);
            titleSegTitleNum.setText((i + 1) + ".");
                                        //设置 titleSegTitleNum 显示的文字
            titleSegTitleNum.setTextColor(Color.BLACK);
                                        //设置 titleSegTitleNum 文字的颜色
            TextPaint xhtp = titleSegTitleNum.getPaint();
            xhtp.setFakeBoldText(true);//设置 titleSegTitleNum 文字粗体显示
            //标题
            titleSegTitle = new TextView(this);         //创建 TextView 组件
            //创建布局参数
            LinearLayout.LayoutParams titleSegParam = new LinearLayout.
            LayoutParams(
                    207, LayoutParams.WRAP_CONTENT);
            titleSegTitle.setLayoutParams(titleSegParam);//titleSegTitle
            添加 titleSegParam 布局参数
            titleSegTitle.setText(route.getName().replaceAll("——", "—") +
            " , "
                    + route.getMileage() + "公里");
            titleSegTitle.setTextColor(Color.BLACK);
                                    //设置 titleSegTitle 文字的颜色
```

```
TextPaint bttp = titleSegTitle.getPaint();
bttp.setFakeBoldText(true);//设置 titleSegTitle 文字粗体显示

//地图按钮
mapButt = new ImageButton(this);        //创建 ImageButton 组件
//创建布局参数
LinearLayout.LayoutParams mapButtParam = new LinearLayout.
LayoutParams(
        50, 23);
mapButtParam.setMargins(5, 0, 0, 0);   //设置边距
mapButt.setLayoutParams(mapButtParam);
                           //mapButt 添加 mapButtParam 布局参数
mapButt.setBackgroundResource(R.drawable.mapbut);
                           //设置 mapButt 显示的图片
mapButt.setOnClickListener(new Button.OnClickListener(){
                           //mapButt 单击事件
            public void onClick(View arg0) {
                Bundle bl = new Bundle();   //实例化 Bundle
                bl.putString("roadId", route.getId());
                                            //保存路书 id
                bl.putString("loc", route.getStartPoint());
                                            //保存起点位置
                bl.putString("loc2", route.getEndPoint());
                                            //保存终点位置
                bl.putString("dis", route.getMileage());
                                            //保存里程
                bl.putBoolean("pan", true);
                bl.putString("tripId", tripId); //保存分段 id
                bl.putString("tripName", TripName);
                                            //保存分段名称
                Intent it = new Intent();   //实例化 Intent
                it.setClass(TripSegment.this, RoadMapView.
                class);                     //设置 Class
                it.putExtras(bl);
                            //将该对象作为参数传递给下一个窗体
                startActivity(it);          //启动 Activity
            }
        });

//将分段标题序号，分段标题和地图按钮添加到该 titleSegLayout 上
titleSegLayout.addView(titleSegTitleNum);
titleSegLayout.addView(titleSegTitle);
titleSegLayout.addView(mapButt);

//分段路书线路介绍
tripDescSeg = new TextView(this);           //实例化 TextView
//创建布局参数
LinearLayout.LayoutParams tripDescSegParam = new LinearLayout.
LayoutParams(
        267, LayoutParams.WRAP_CONTENT);
tripDescSegParam.setMargins(18, 6, 0, 8);  //设置边距
//为 tripDescSeg 添加布局参数 tripDescSegParam
tripDescSeg.setLayoutParams(tripDescSegParam);
tripDescSeg.setText(route.getDesc().replace("##", "\n\n"));
                               //分段描述中的##，显示时替换为换行
```

```
        tripDescSeg.setTextColor(Color.BLACK);      //设置文字颜色
        tripDescSeg.setLineSpacing(1.0f, 1.1f);     //设置文字间距
        //分段路数图片
        tripPicSeg = new ImageView(this);           //实例化 ImageView
        //创建布局参数
        LinearLayout.LayoutParams tripPicSegParam = new LinearLayout.
        LayoutParams(
                LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
        tripPicSegParam.setMargins(25, 0, 0, 20);  //设置边距
        tripPicSeg.setLayoutParams(tripPicSegParam);
                            //为 tripPicSeg 添加布局参数 tripPicSegParam

        //读图片文件并显示
        String tripImg = route.getImage().split("[.]")[0] + "_b.jpg";
        InputStream iso;
        try {
            iso = this.getAssets().open(
                    tripId + "/SmallRoute" + "/" + route.getId()
                            + "/images/" + tripImg); //打开 assets 下的图片
            Bitmap bitmap = null;
            bitmap = BitmapFactory.decodeStream(iso);
            tripPicSeg.setImageBitmap(bitmap);
        } catch (IOException e1) {
            //TODO Auto-generated catch block
            e1.printStackTrace();
        }

        //tripPicSeg 图片单击事件
        tripPicSeg.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                Intent it = new Intent();
                Bundle tripDetailPic = new Bundle();//创建 Bundle 对象
                it.setClass(TripSegment.this, TripDetailPic.class);
                                                    //设置 Class 参数
                //图片路径
                String tripImg = tripId + "/SmallRoute" + "/"
                        + route.getId() + "/images/" + "y "
                        + route.getImage().split("[.]")[0] + ".jpg";
                tripDetailPic.putString("tripImgPath", tripImg);
                it.putExtras(tripDetailPic);
                startActivity(it);                  //启动窗体
            }

        });

        //将动态生成的添加到 layout 上
        tripSegsLayout.addView(titleSegLayout);
        tripSegsLayout.addView(tripDescSeg);
        tripSegsLayout.addView(tripPicSeg);
    }
```

路书分段列表窗体运行效果如图 13.22 所示。

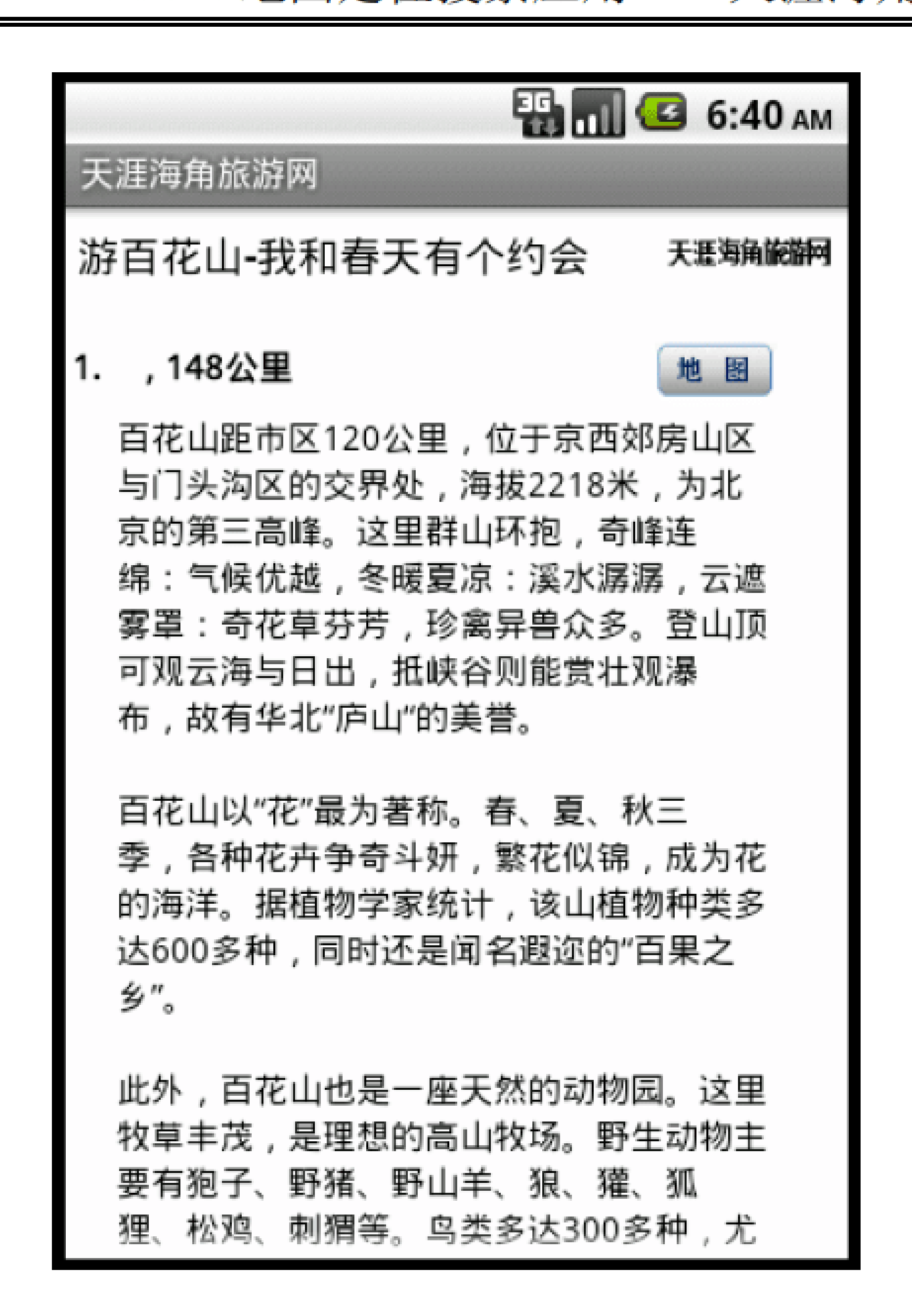


图 13.22　线路分段详情页展示

13.15　地图窗体类的设计及实现

单击图 13.7 中的“地图”按钮，在地图上显示线路信息及兴趣点信息。

该类模块主要包括地图线路展示、地图兴趣点展示、GPS 卫星定位、兴趣点接近播报等功能模块。

13.15.1　线路展示

线路信息保存在 trackList 集合中，以两点间画线的方式连接，形成展示的线路走向。因为需要连线的点数量很多，有可能造成地图拖曳反应缓慢，所以线路描绘时作了处理：当用户触摸屏幕并拖曳地图时，不重新绘制线段；只有在拖曳结束后，才开始重新绘制。

首先要准备一个内部类，负责线段的描绘，代码如下：

```
//内部类，线段
public class Line extends Overlay {

    private List<GeoPoint> points;                    //定义 List 对象
    private Paint paint;                              //定义 Paint 对象
    private boolean pan = true;                       //定义 pan 变量
    private int count = 0;                            //定义 count 变量

    public Line(List<GeoPoint> points) {
        this.points = points;
        paint = new Paint();                          //实例化 paint 对象

        paint.setARGB(250, 249, 105, 8);              //设置 ARGB
```

```
            paint.setAntiAlias(true);                        //设置 AntiAlias
            paint.setStyle(Paint.Style.FILL_AND_STROKE); //设置 Style
            paint.setStrokeWidth(4);                        //设置 StrokeWidth
        }

        public Line(List<GeoPoint> points, Paint paint) {
            this.points = points;
            this.paint = paint;
        }

        //单击触发
        public boolean onTouchEvent(MotionEvent e, MapView mapView) {
            if (e.getAction() == MotionEvent.ACTION_DOWN) {
                pan = false;
            } else if (e.getAction() == MotionEvent.ACTION UP) {
                pan = true;
            }
            return false;
        }

        public void draw(Canvas canvas, MapView mapView, boolean shadow) {
            count = count + 1;                          //累加 count 变量
            if (pan) {
                if (count % 2 == 0) {
                    if (!shadow) {
                        Projection projection = mapView.getProjection();
                                                    //获取 Projection 对象
                        if (points != null) {
                            if (points.size() >= 2) {
                                Point start = projection.toPixels(
                                        points.get(0), null);
                                                    //获取前 Point 对象
                                for (int i = 1; i < points.size(); i++) {
                                    Point end = projection.toPixels(points
                                            .get(i), null);
                                                    //获取后 Point 对象
                                    canvas.drawLine(start.x, start.y,
                                    end.x, end.y, paint);  //两点间画线
                                    start = end;    //后点变前点
                                }
                            }
                        }
                    }
                }
            }
        }
    }
```

然后一个调用内部类画线的方法，代码如下：

```
    //准备数据开始画线
    public void preToLine() {
        try {
            List<GeoPoint> points = new ArrayList<GeoPoint>();
                                                    //实例化 points 对象
            for (int i = 0; i < trackList.size(); i += 1) {
                                                    //循环 trackList
                String loc = trackList.get(i).getTrackPoints(); //获取坐标
                if (!"".equals(loc)) {
```

```
                    String str[] = loc.split(",");            //拆分坐标
                    for (int j = 0; j < str.length; j += 1) { //循环坐标段
                        String stemp[] = str[j].split(" ");  //拆分坐标点
                        GeoPoint point = new GeoPoint((int) (Double
                                .valueOf(stemp[0]) * 1E6), (int) (Double
                                .valueOf(stemp[1]) * 1E6));
                                                    //实例化 GeoPoint 对象
                        points.add(point);          //添加坐标点
                    }
                }
            }

            line = new Line(points);                //实例化 line 对象
            map.getOverlays().add(line);            //添加 line 到地图
            map.invalidate();                       //更新地图
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

我们只需要在获取线段数据资料后调用，调用方法即可，代码如下：

```
    //获取路书线路信息
    public void getMapLine() {
        myDialog = ProgressDialog.show(RoadMapView.this, "请稍等...", "执
        行运算中...",true);                         //实例化 myDialog
        new Thread() {                              //开启新的线程
            public void run() {
                try {
                    //实例化 trackList
                    trackList = new WAnalysisFile().getTrackPointList-
                    ByRouteId(
                            RoadMapView.this, roadId, tripId);
                    Message m = new Message();      //实例化 Message 对象
                    RoadMapView.this.lineHandler.sendMessage(m);
                                                    //调用 lineHandler
                } catch (Exception e) {
                    e.printStackTrace();
                } finally {
                    myDialog.dismiss();
                }
            }
        }.start();
    }

    //线路 Handler
    Handler lineHandler = new Handler() {  //定义 Handler 对象 lineHandler
        public void handleMessage(Message msg) {
            preToLine();
            getMapPois();
        }
    };
```

13.15.2　兴趣点展示

兴趣点信息保存在 poiList 集合中，以地图标注的形式显示在地图上。当用户单击某一

个兴趣点标注时，在相应位置描绘提示信息框。用户单击提示信息框后，程序切换到兴趣点详情窗体展示该兴趣点详情。用户可以在菜单中选择在地图上显示或不显示兴趣点标注。

首先创建一个内部类，负责兴趣点图标的显示，以及对点事件的响应，代码如下：

```
//内部类，标记
public class Marker extends ItemizedOverlay<OverlayItem> {
    //实例化 markerList
    private ArrayList<OverlayItem> markerList = new ArrayList<Overlay-
    Item>();

    public Marker(Drawable defaultMarker) {
        super(boundCenterBottom(defaultMarker));
    }

    @Override
    protected OverlayItem createItem(int arg0) {
        //获取某个 markerList 中元素
        return markerList.get(arg0);
    }

    @Override
    public int size() {
        //获取 markerList 大小
        return markerList.size();
    }

    public void addOverlay(OverlayItem overlay) {
        markerList.add(overlay);                      //添加元素
        populate();                                   //调用初始化
    }

    protected boolean onTap(int i) {

        if (createItem(i).getTitle().equals("poiinfo")) {
            geoPoi(createItem(i).getSnippet());       //调用页面切换
            return false;
        }

        if (to != null) {
            map.getOverlays().remove(to);             //移除显示框
        }

        poiId = createItem(i).getSnippet();           //获取 poiId
        String str = createItem(i).getTitle();        //获取标题
        if (str.length() > 8) {
            str = str.substring(0, 4) + "..."
                    + str.substring(str.length() - 1, str.length());
                                                      //字符截取
        }
        to = new TextOverlay(str, createItem(i).getPoint());
                                                      //实例化显示框
        map.getOverlays().add(to);                    //添加显示框到地图

        return false;
    }

}
```

然后准备一个获取兴趣点资料的方法，代码如下：

```
    //获取兴趣点信息
    public void getMapPois() {
        myDialog = ProgressDialog.show(RoadMapView.this, "请稍等...", "执
        行运算中...", true);                              //实例化 myDialog 对象
        new Thread() {                                   //开启新的线程
            public void run() {
                try {
                    //获取 poiList 对象
                    poiList = new WAnalysisFile().getSmallPoiPointList(
                            RoadMapView.this, roadId, tripId);
                    mp3List = new ArrayList<Mp3Point>();//实例化 mp3List 对象
                    for (int i = 0; i < poiList.size(); i += 1) {//循环 poiList
                        PoiPoint poi = poiList.get(i);
                        if (!"".equals(poi.getMp3Path())) {
                                                        //判断是否有 mp3 属性
                            Mp3Point mp3 = new Mp3Point();
                                                        //实例化Mp3Point 对象
                            mp3.setLat(poi.getLat());   //设置坐标 lat 值
                            mp3.setLon(poi.getLon());   //设置坐标 lon 值
                            mp3.setMp3Id(poi.getMp3Id());  //设置 Mp3Id
                            mp3.setMp3Path(poi.getMp3Path());
                                                        //设置 Mp3Path
                            mp3.setPan(true);           //设置是否播放
                            mp3.setMp3Range(poi.getMp3Range());
                                                        //设置 Range
                            mp3List.add(mp3);           //添加对象到 mp3List
                        }
                    }

                    Message m = new Message();          //实例化 Message 对象
                    RoadMapView.this.poiHandler.sendMessage(m);
                                                        //调用 poiHandler
                } catch (Exception e) {
                    e.printStackTrace();
                } finally {
                    myDialog.dismiss();
                }
            }
        }.start();
    }

    //兴趣点 Handler
    Handler poiHandler = new Handler() {   //实例化 Handler 对象 poiHandler
        public void handleMessage(Message msg) {
            preToPoiMarker();
            showIco();
        }
    };
```

因为兴趣点有多种类别，分别以不同的图符显示在地图上，所以在获取兴趣点资料后，要分别以不同的图片来实例化兴趣点，代码如下：

```
//准备数据开始标记兴趣点
public void preToPoiMarker() {
    for (int i = 0; i < poiList.size(); i += 1) {  //循环 poiList
```

```
        PoiPoint poi = poiList.get(i);                      //实例化PoiPoint对象
        if ("Forest".equals(poi.getCategoryId())            //判断类别
                || "Park".equals(poi.getCategoryId())) {
            if (marker1 == null) {
                Drawable drawable = this.getResources().getDrawable(
                        R.drawable.t1);                     //实例化Drawable对象
                marker1 = new Marker(drawable);             //实例化 marker1
            }
            //实例化 GeoPoint 对象
            GeoPoint gpoint = new GeoPoint((int) (Double.valueOf(poi
                    .getLat()) * 1E6),
                    (int) (Double.valueOf(poi.getLon()) * 1E6));
            //实例化 OverlayItem 对象
            OverlayItem overlayitem = new OverlayItem(gpoint,
                    poi.getName(), poi.getId());
            //添加 overlayitem 到 marker1
            marker1.addOverlay(overlayitem);
        } else if ("Museum".equals(poi.getCategoryId()) //判断类别
                || "Letter B, Red".equals(poi.getCategoryId())
                || "Flag, Red".equals(poi.getCategoryId())
                || "Church".equals(poi.getCategoryId())
                || "Letter A, Red".equals(poi.getCategoryId())) {
            if (marker2 == null) {
                Drawable drawable = this.getResources().getDrawable(
                        R.drawable.t2);                     //实例化Drawable对象
                marker2 = new Marker(drawable);             //实例化 marker1
            }
            //实例化 GeoPoint 对象
            GeoPoint gpoint = new GeoPoint((int) (Double.valueOf(poi
                    .getLat()) * 1E6),
                    (int) (Double.valueOf(poi.getLon()) * 1E6));
            //实例化 OverlayItem 对象
            OverlayItem overlayitem = new OverlayItem(gpoint,
                    poi.getName(), poi.getId());
            //添加 overlayitem 到 marker2
            marker2.addOverlay(overlayitem);
        } else if ("Zoo".equals(poi.getCategoryId())    //判断类别
                || "Crossing".equals(poi.getCategoryId())
                || "Marina".equals(poi.getCategoryId())
                || "City (Large)".equals(poi.getCategoryId())
                || "City (Medium)".equals(poi.getCategoryId())
                || "City (Small)".equals(poi.getCategoryId())
                || "Letter C, Green".equals(poi.getCategoryId())
                || "Horn".equals(poi.getCategoryId())) {
            if (marker3 == null) {
                Drawable drawable = this.getResources().getDrawable(
                        R.drawable.t3);                     //实例化Drawable对象
                marker3 = new Marker(drawable);             //实例化 marker1
            }
            //实例化 GeoPoint 对象
            GeoPoint gpoint = new GeoPoint((int) (Double.valueOf(poi
                    .getLat()) * 1E6),
                    (int) (Double.valueOf(poi.getLon()) * 1E6));
            //实例化 OverlayItem 对象
            OverlayItem overlayitem = new OverlayItem(gpoint,
                    poi.getName(), poi.getId());
            //添加 overlayitem 到 marker3
            marker3.addOverlay(overlayitem);
```

```
        } else if ("Shopping Center".equals(poi.getCategoryId())) {
                                                        //判断类别
            if (marker4 == null) {
                Drawable drawable = this.getResources().getDrawable(
                        R.drawable.t4);                 //实例化 Drawable 对象
                marker4 = new Marker(drawable);         //实例化 marker1
            }
            //实例化 GeoPoint 对象
            GeoPoint gpoint = new GeoPoint((int) (Double.valueOf(poi
                    .getLat()) * 1E6),
                    (int) (Double.valueOf(poi.getLon()) * 1E6));
            //实例化 OverlayItem 对象
            OverlayItem overlayitem = new OverlayItem(gpoint,
                    poi.getName(), poi.getId());
            //添加 overlayitem 到 marker4
            marker4.addOverlay(overlayitem);
        } else if ("Scenic Area".equals(poi.getCategoryId())) {//判断类别
            if (marker5 == null) {
                Drawable drawable = this.getResources().getDrawable(
                        R.drawable.t5);                 //实例化 Drawable 对象
                marker5 = new Marker(drawable);         //实例化 marker1
            }
            //实例化 GeoPoint 对象
            GeoPoint gpoint = new GeoPoint((int) (Double.valueOf(poi
                    .getLat()) * 1E6),
                    (int) (Double.valueOf(poi.getLon()) * 1E6));
            //实例化 OverlayItem 对象
            OverlayItem overlayitem = new OverlayItem(gpoint,
                    poi.getName(), poi.getId());
            //添加 overlayitem 到 marker5
            marker5.addOverlay(overlayitem);
        } else if ("Gas Station".equals(poi.getCategoryId())) {//判断类别
            if (marker6 == null) {
                Drawable drawable = this.getResources().getDrawable(
                        R.drawable.t6);                 //实例化 Drawable 对象
                marker6 = new Marker(drawable);         //实例化 marker1
            }
            //实例化 GeoPoint 对象
            GeoPoint gpoint = new GeoPoint((int) (Double.valueOf(poi
                    .getLat()) * 1E6),
                    (int) (Double.valueOf(poi.getLon()) * 1E6));
            //实例化 OverlayItem 对象
            OverlayItem overlayitem = new OverlayItem(gpoint,
                    poi.getName(), poi.getId());
            //添加 overlayitem 到 marker6
            marker6.addOverlay(overlayitem);
        } else if ("Lodging".equals(poi.getCategoryId())) {    //判断类别
            if (marker7 == null) {
                Drawable drawable = this.getResources().getDrawable(
                        R.drawable.t7);                 //实例化 Drawable 对象
                marker7 = new Marker(drawable);         //实例化 marker1
            }
            //实例化 GeoPoint 对象
            GeoPoint gpoint = new GeoPoint((int) (Double.valueOf(poi
                    .getLat()) * 1E6),
                    (int) (Double.valueOf(poi.getLon()) * 1E6));
            //实例化 OverlayItem 对象
```

```
            OverlayItem overlayitem = new OverlayItem(gpoint,
                    poi.getName(), poi.getId());
            //添加 overlayitem 到 marker7
            marker7.addOverlay(overlayitem);
        } else if ("Restaurant".equals(poi.getCategoryId())) {//判断类别
            if (marker8 == null) {
                Drawable drawable = this.getResources().getDrawable(
                        R.drawable.t8);             //实例化 Drawable 对象
                marker8 = new Marker(drawable);     //实例化 marker1
            }
            //实例化 GeoPoint 对象
            GeoPoint gpoint = new GeoPoint((int) (Double.valueOf(poi
                    .getLat()) * 1E6),
                    (int) (Double.valueOf(poi.getLon()) * 1E6));
            //实例化 OverlayItem 对象
            OverlayItem overlayitem = new OverlayItem(gpoint,
                    poi.getName(), poi.getId());
            //添加 overlayitem 到 marker8
            marker8.addOverlay(overlayitem);
        } else if ("Waypoint".equals(poi.getCategoryId())) {   //判断类别
            if (marker9 == null) {
                Drawable drawable = this.getResources().getDrawable(
                        R.drawable.t9);             //实例化 Drawable 对象
                marker9 = new Marker(drawable);     //实例化 marker1
            }
            //实例化 GeoPoint 对象
            GeoPoint gpoint = new GeoPoint((int) (Double.valueOf(poi
                    .getLat()) * 1E6),
                    (int) (Double.valueOf(poi.getLon()) * 1E6));
            //实例化 OverlayItem 对象
            OverlayItem overlayitem = new OverlayItem(gpoint,
                    poi.getName(), poi.getId());
            //添加 overlayitem 到 marker9
            marker9.addOverlay(overlayitem);
        } else if ("Toll Booth".equals(poi.getCategoryId())) {//判断类别
            if (marker10 == null) {
                Drawable drawable = this.getResources().getDrawable(
                        R.drawable.t10);            //实例化 Drawable 对象
                marker10 = new Marker(drawable);    //实例化 marker1
            }
            //实例化 GeoPoint 对象
            GeoPoint gpoint = new GeoPoint((int) (Double.valueOf(poi
                    .getLat()) * 1E6),
                    (int) (Double.valueOf(poi.getLon()) * 1E6));
            //实例化 OverlayItem 对象
            OverlayItem overlayitem = new OverlayItem(gpoint,
                    poi.getName(), poi.getId());
            //添加 overlayitem 到 marker10
            marker10.addOverlay(overlayitem);
        } else if ("Parking Area".equals(poi.getCategoryId())) {//判断类别
            if (marker11 == null) {
                Drawable drawable = this.getResources().getDrawable(
                        R.drawable.t11);            //实例化 Drawable 对象
                marker11 = new Marker(drawable);    //实例化 marker1
            }
            //实例化 GeoPoint 对象
            GeoPoint gpoint = new GeoPoint((int) (Double.valueOf(poi
```

```
                .getLat()) * 1E6),
                (int) (Double.valueOf(poi.getLon()) * 1E6));
        //实例化 OverlayItem 对象
        OverlayItem overlayitem = new OverlayItem(gpoint,
                poi.getName(), poi.getId());
        //添加 overlayitem 到 marker11
        marker11.addOverlay(overlayitem);
    } else {//判断默认类别
        if (marker0 == null) {
            Drawable drawable = this.getResources().getDrawable(
                    R.drawable.mountain);       //实例化 Drawable 对象
            marker0 = new Marker(drawable);     //实例化 marker1
        }
        //实例化 GeoPoint 对象
        GeoPoint gpoint = new GeoPoint((int) (Double.valueOf(poi
                .getLat()) * 1E6),
                (int) (Double.valueOf(poi.getLon()) * 1E6));
        //实例化 OverlayItem 对象
        OverlayItem overlayitem = new OverlayItem(gpoint,
                poi.getName(), poi.getId());
        //添加 overlayitem 到 marker0
        marker0.addOverlay(overlayitem);
    }
  }

}
```

当单击兴趣点图标后，会显示一个简单的提示窗口，用户在单击这个窗口后，程序会切换到兴趣点详情界面。为了实现这一效果，首先要准备一个描绘提示窗口的内部类，代码如下：

```
//文字提示框
public class TextOverlay extends Overlay {
    private String str;                                  //定义 str 对象
    private GeoPoint p;                                  //定义 GeoPoint 对象

    public TextOverlay(String str, GeoPoint p) {
        this.str = str;
        this.p = p;
    }

    public boolean draw(Canvas arg0, MapView arg1, boolean arg2, long arg3) {
        if (!arg2) {
            Paint paint = new Paint();                   //实例化 Paint 对象
            paint.setTextSize(13);                       //设置 TextSize
            paint.setColor(Color.WHITE);                 //设置 Color
            paint.setAntiAlias(true);                    //设置 AntiAlias
            paint.setFakeBoldText(true);                 //设置 FakeBoldText
            Paint bpaint = new Paint();                  //实例化 Paint 对象
            bpaint.setColor(Color.BLACK);                //设置 Color
            bpaint.setAntiAlias(true);                   //设置 AntiAlias
            bpaint.setAlpha(150);                        //设置 Alpha

            Point temp = map.getProjection().toPixels(p, null);
                                                         //实例化 Point 对象
            int x1 = temp.x + 15;                        //计算对象 x 坐标
```

```
            int y1 = temp.y - 30;                            //计算对象 y 坐标

            int count = str.length();                        //显示文字个数
            int x2 = x1 + (count + 3) * 13;                  //计算所需长度
            RectF boval = new RectF(x1, y1, x2, y1 + 30);  //定义圆角矩形
            setXY(x1, y1 + 50, x2, y1 + 80);                 //保存区域范围

            arg0.drawRoundRect(boval, 10, 10, bpaint);       //绘制圆角矩形
            arg0.drawText(str + "  >", x1 + 13, y1 + 20, paint); //绘制文字

        }
        return super.draw(arg0, arg1, arg2, arg3);
    }

}
```

为了判断用户是否单击了提示窗口，我们将提示窗口 4 个角的坐标保存在数组变量中，然后同用户单击的屏幕坐标进行比对。如果单击位置不在提示窗口的范围内，则取消提示窗口，否则就切换到选中兴趣点的详情界面，代码如下：

```
//获取详情框坐标像素点
public void setXY(int x1, int y1, int x2, int y2) {
    xy[0] = x1;
    xy[1] = x2;
    xy[2] = y1;
    xy[3] = y2;
}

//取消详情框
public boolean dispatchTouchEvent(MotionEvent ev) {

    int cx = (int) ev.getX();                             //获取屏幕单击 x 坐标
    int cy = (int) ev.getY();                             //获取屏幕单击一坐标
    //判断单击位置范围
    if ((cx < xy[0] || cx > xy[1]) || (cy < xy[2] || cy > xy[3])) {
        if (to != null) {                                 //移除单击窗口
            map.getOverlays().remove(to);
            to = null;
            poiId = "";
            map.invalidate();
        }
    } else {
        if (!"".equals(poiId)) {
            geoPoi(poiId);                                //切换屏幕
        }
    }
    return super.dispatchTouchEvent(ev);
}
```

切换到兴趣点详情界面的代码如下：

```
//获取兴趣点详情
private void geoPoi(String temp) {
    Bundle bl = new Bundle();                             //实例化 Bundle 对象
    bl.putSerializable("poiObj", (Serializable) getPoi(temp));
                                                          //插入 PoiPoint 数据
    bl.putString("tripId", tripId);                       //插入 tripId
```

```
    bl.putString("tripName", TripName);             //插入 tripName
    Intent it = new Intent();                       //实例化 Intent
    it.setClass(RoadMapView.this, PoiDetail.class); //设置 Class
    it.putExtras(bl);                               //设置 Extras
    startActivity(it);                              //启动 Activity
}

private PoiPoint getPoi(String id) {
    PoiPoint poi = new PoiPoint();                  //实例化 PoiPoint 对象
    for (int i = 0; i < poiList.size(); i += 1) {   //循环 poiList
        if (poiList.get(i).getId().equals(id)) {    //判断符合条件
            poi = poiList.get(i);                   //获取 PoiPoint 值
            break;
        }
    }
    return poi;
}
```

最后就是对兴趣点图标的显示和隐藏操作，代码如下：

```
//显示图标
public void showIco() {
    if (marker0 != null) {
        map.getOverlays().add(marker0);         //在地图上添加 marker0
    }
    if (marker1 != null) {
        map.getOverlays().add(marker1);         //在地图上添加 marker1
    }
    if (marker2 != null) {
        map.getOverlays().add(marker2);         //在地图上添加 marker2
    }
    if (marker3 != null) {
        map.getOverlays().add(marker3);         //在地图上添加 marker3
    }
    if (marker4 != null) {
        map.getOverlays().add(marker4);         //在地图上添加 marker4
    }
    if (marker5 != null) {
        map.getOverlays().add(marker5);         //在地图上添加 marker5
    }
    if (marker6 != null) {
        map.getOverlays().add(marker6);         //在地图上添加 marker6
    }
    if (marker7 != null) {
        map.getOverlays().add(marker7);         //在地图上添加 marker7
    }
    if (marker8 != null) {
        map.getOverlays().add(marker8);         //在地图上添加 marker8
    }
    if (marker9 != null) {
        map.getOverlays().add(marker9);         //在地图上添加 marker9
    }
    if (marker10 != null) {
        map.getOverlays().add(marker10);        //在地图上添加 marker10
    }
    if (marker11 != null) {
        map.getOverlays().add(marker11);        //在地图上添加 marker11
    }
```

```
    map.invalidate();
}

//隐藏图标
public void hideIco() {
    if (marker0 != null) {
        map.getOverlays().remove(marker0);     //在地图上移除 marker0
    }
    if (marker1 != null) {
        map.getOverlays().remove(marker1);     //在地图上移除 marker1
    }
    if (marker2 != null) {
        map.getOverlays().remove(marker2);     //在地图上移除 marker2
    }
    if (marker3 != null) {
        map.getOverlays().remove(marker3);     //在地图上移除 marker3
    }
    if (marker4 != null) {
        map.getOverlays().remove(marker4);     //在地图上移除 marker4
    }
    if (marker5 != null) {
        map.getOverlays().remove(marker5);     //在地图上移除 marker5
    }
    if (marker6 != null) {
        map.getOverlays().remove(marker6);     //在地图上移除 marker6
    }
    if (marker7 != null) {
        map.getOverlays().remove(marker7);     //在地图上移除 marker7
    }
    if (marker8 != null) {
        map.getOverlays().remove(marker8);     //在地图上移除 marker8
    }
    if (marker9 != null) {
        map.getOverlays().remove(marker9);     //在地图上移除 marker9
    }
    if (marker10 != null) {
        map.getOverlays().remove(marker10);    //在地图上移除 marker10
    }
    if (marker11 != null) {
        map.getOverlays().remove(marker11);    //在地图上移除 marker11
    }
    map.invalidate();
}
```

13.15.3　GPS 卫星定位

调用手机的 GPS 卫星定位模块，每隔一段时间检查一下手机当前位置，并以地图标注的形式显示出来。用户可以通过菜单选择是否在地图上展示当前位置。

要实现定位功能，首先要启动位置监听，代码如下：

```
//位置监听器
private final LocationListener locationListener = new LocationListener(){
                                                        //实例化位置监听器

    public void onLocationChanged(Location location) {//档位置改变时触发
        if (panLoc) {
```

```
            showNowLoc();                                   //显示位置
            preMp3List(location);                           //播放 MP3
        }
    }

    public void onProviderDisabled(String provider) {

    }

    public void onProviderEnabled(String provider) {

    }

    public void onStatusChanged(String provider, int status, Bundle extras)
    {

    }
};
```

然后准备显示当前位置的方法，代码如下：

```
//显示当前位置
public void showNowLoc() {

    Criteria criteria = new Criteria();                 //定义 Criteria 对象
    criteria.setAccuracy(Criteria.ACCURACY FINE);       //设置 Accuracy
    criteria.setAltitudeRequired(false);                //设置 AltitudeRequired
    criteria.setBearingRequired(false);                 //设置 BearingRequired
    criteria.setCostAllowed(false);                     //设置 CostAllowed
    criteria.setPowerRequirement(Criteria.POWER_LOW);
                                                        //设置 PowerRequirement

    Location location = locationManager
        .getLastKnownLocation(locationManager.getBestProvider(criteria,
         true));                                        //实例化 Location

    if (location == null) {                             //判断信号
        Toast.makeText(RoadMapView.this, "没有 GPS 信号", Toast.LENGTH
        SHORT)
                .show();
    } else {
        panLoc = true;
        double trueLat = location.getLatitude();        //获取坐标 Lat 值
        double trueLon = location.getLongitude();       //获取坐标 Lon 值
        if (loc != null) {
            map.getOverlays().remove(loc);              //移除位置图标
            map.invalidate();                           //更新地图
        }
        Drawable drawable = this.getResources().getDrawable(
                R.drawable.weizhi);                     //实例化 Drawable 对象
        loc = new Marker(drawable);
        //实例化 GeoPoint
        GeoPoint point = new GeoPoint((int) (trueLat * 1E6),
                (int) (trueLon * 1E6));
        //实例化 OverlayItem
        OverlayItem overlayitem = new OverlayItem(point, "", "");
        loc.addOverlay(overlayitem);
        map.getOverlays().add(loc);                     //添加 loc 到地图
```

```
        map.invalidate();                           //更新地图
    }
    
}
```

13.15.4　兴趣点接近播报

该功能依靠 GPS 卫星定位功能。兴趣点集合中有一些特殊的兴趣点带有 MP3 资料，当定位发现当前位置离这些兴趣点的距离达到设定标准的时候，调用手机的 MP3 播放功能，播放音频资料。

首先要准备需要播放的 mp3 的列表，代码如下：

```
//准备播放 mp3 列表
public void preMp3List(Location location) {
    double trueLat = location.getLatitude();          //获取坐标 Lat 值
    double trueLon = location.getLongitude();         //获取坐标 Lon 值
    for (int i = 0; i < mp3List.size(); i += 1) {     //循环 mp3List
        Mp3Point mp3 = mp3List.get(i);                //获取 Mp3Point 对象
        if (mp3.isPan()) {
            float[] results = new float[1];
            //获取两点间距离
            Location.distanceBetween(trueLat, trueLon, Double.valueOf(mp3
                    .getLat()), Double.valueOf(mp3.getLon()), results);
            int dis = (int) results[0];
            if (dis <= mp3.getMp3Range()) {           //判断距离值
                mp3.setPan(false);                    //设置播放状态
                playerList.add(mp3);                  //加入播放列表
                if (!player.isPlaying()) {            //判断播放状态
                    playMp3();
                }
                break;
            }
        }
    }
}
```

然后在定位确认距离接近后，进行播放，代码如下：

```
//开始播放 mp3 序列
public void playMp3() {
    try {
        Mp3Point mp3 = playerList.poll();             //获取播放对象
        if (mp3 != null) {
            player.reset();                           //重置播放器
            //获取播放资源
            AssetFileDescriptor afd = RoadMapView.this.getAssets().openFd(
                    tripId + "/SmallRoute" + "/" + roadId + "/mp3/"
                            + mp3.getMp3Path().toLowerCase());
            //设置播放资源
            player.setDataSource(afd.getFileDescriptor(), afd
                    .getStartOffset(), afd.getLength());
            player.prepare();                         //播放器准备
            player.start();                           //开始播放
        }
    } catch (Exception e) {
```

```
        e.printStackTrace();
    }
}
```

音频播放器和 GPS 卫星定位监听器需要在该功能界面结束时撤销，以免影响其他功能模块的展示，代码如下：

```
//撤销关闭 MP3
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == 4 && player.isPlaying()) {                //判断按键
        player.reset();                                      //重置播放器
    }
    return super.onKeyDown(keyCode, event);
}

//关闭位置监听
protected void onStop() {
    locationManager.removeUpdates(locationListener);         //移除位置监听器
    super.onStop();
}

//打开位置监听
protected void onRestart() {
    //重置位置监听器
    locationManager = (LocationManager) getSystemService(Context.
    LOCATION_SERVICE);
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
            1000, 0.0001f, locationListener);
    super.onRestart();
}
```

13.15.5　菜单功能

在菜单项上，为用户准备了一些操作功能，如兴趣点的显示隐藏、当前位置的显示隐藏等，代码如下：

```
//地图页面菜单
public boolean onCreateOptionsMenu(Menu menu) {

    MenuItem mnuAbout = menu.add(0, 0, 0, "返回");        //定义 MenuItem 对象
    mnuAbout.setIcon(R.drawable.aboutmenu);               //设置图标

    MenuItem mnuHome = menu.add(0, 1, 1, "隐藏兴趣点");   //定义 MenuItem 对象
    mnuHome.setIcon(R.drawable.homemenu);                 //设置图标

    MenuItem mnuFanhui = menu.add(0, 2, 2, "显示当前位置"); //定义 MenuItem 对象
    mnuFanhui.setIcon(R.drawable.fanhuimenu);             //设置图标

    return super.onCreateOptionsMenu(menu);
}

//菜单响应事件
public boolean onOptionsItemSelected(MenuItem item) {
```

```
    super.onOptionsItemSelected(item);
    switch (item.getItemId()) {
    case 0:
        RoadMapView.this.finish();                    //结束当前 MapActivity
        break;
    case 1:
        if ("隐藏兴趣点".equals(item.getTitle().toString())) {//判断菜单文字
            item.setTitle("显示兴趣点");              //设置菜单文字
            hideIco();//隐藏图标
        } else {
            item.setTitle("隐藏兴趣点");              //设置菜单文字
            showIco();//显示图标
        }
        break;
    case 2:
        if ("显示当前位置".equals(item.getTitle().toString())) {
                                                      //判断菜单文字
            showNowLoc();                             //显示位置
            if (panLoc) {
                item.setTitle("隐藏当前位置");        //设置菜单文字
            }
        } else {
            panLoc = false;
            item.setTitle("显示当前位置");            //设置菜单文字
            map.getOverlays().remove(loc);            //移除位置图标
            map.invalidate();                         //更新地图
            if (player.isPlaying()) {                 //判断播放器状态
                player.reset();                       //重置播放器
            }
        }
        break;
    }
    return true;
}
```

13.15.6　地图功能的初始化准备

为了让地图界面的各种功能可以正常运行，我们需要在 onCreate()方法中进行一些初始化的操作，代码如下：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.roadmapview);             //加载 XML 配置文件
    this.setTitle("天涯海角旅游网");                  //设置标题

    Bundle bl = this.getIntent().getExtras();         //获取 Bundle 对象
    TripName = bl.getString("tripName");              //获取 TripName 的值
    tripId = bl.getString("tripId");                  //获取 tripId 的值
    String loc[] = bl.getString("loc").split(" "); //获取 loc 的值
    pan = bl.getBoolean("pan");                       //获取 pan 的值

    GeoPoint center = null;                           //定义 GeoPoint 对象
    if (pan) {                                        //中心点条件判断
        String loc2[] = bl.getString("loc2").split(" ");  //获取 loc2 的值
        double lat = (Double.valueOf(loc[0]) + Double.valueOf(loc2[0]))/2;
```

```
                                                    //计算中心点 lat 值
        double lon = (Double.valueOf(loc[1]) + Double.valueOf(loc2[1])) /2;
                                                    //计算中心点 lon 值
        center = new GeoPoint((int) (lat * 1E6), (int) (lon * 1E6));
                                                    //设置中心点坐标
    } else {
        center = new GeoPoint((int) (Double.valueOf(loc[0]) * 1E6),
                (int) (Double.valueOf(loc[1]) * 1E6)); //设置中心点坐标
    }

    map = (MapView) findViewById(R.id.tmap);            //实例化 map 对象
    map.setSatellite(false);                            //设置 Satellite
    map.setStreetView(false);                           //设置 StreetView
    map.setBuiltInZoomControls(true);                   //设置地图控件

    MapController mcontrol = map.getController(); //实例化 MapController 对象
    mcontrol.setCenter(center);                         //地图控件添加
    //实例化 locationManager 对象
    locationManager = (LocationManager) getSystemService(Context.
    LOCATION_SERVICE);
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
            1000, 0.0001f, locationListener);  //设置 locationManager 对象

    player.setOnCompletionListener(new OnCompletionListener() {
                                              //设置 MediaPlayer 完成监听
                public void onCompletion(MediaPlayer mp) {
                                              //当播放完毕时执行的方法
                    playMp3();
                }
            });

    double dis = Double.valueOf(bl.getString("dis")); //获取 dis 的值
    if (dis > 200) {
        mcontrol.setZoom(8);                            //设置地图缩放级别
    } else {
        mcontrol.setZoom(10);                           //设置地图缩放级别
    }
    roadId = bl.getString("roadId");                    //获取 roadId 的值
    getMapLine();

}
```

运行效果如图 13.23 所示。

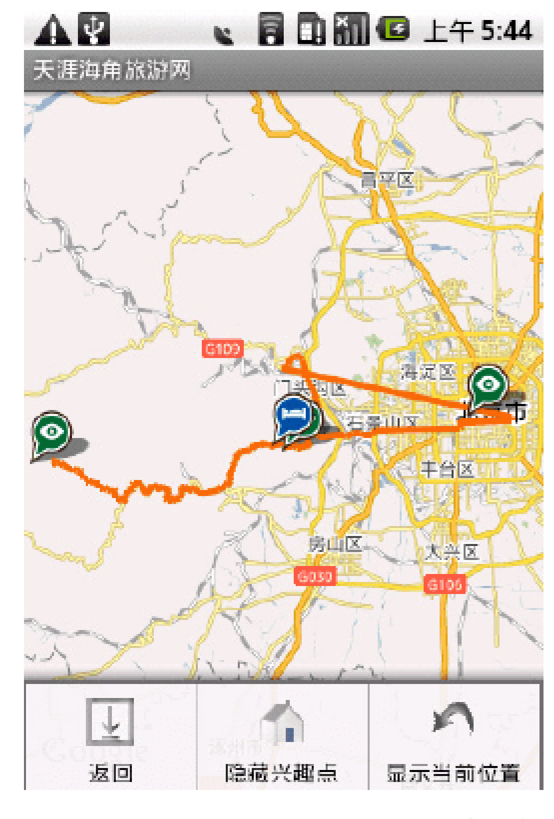


图 13.23　地图展示线路

单击地图上的兴趣点图片，提示兴趣点名称，单击兴趣点名称进入兴趣点详情展示。

13.16　兴趣点列表窗体类的设计及实现

在图 13.4 中单击“兴趣点”按钮，出现兴趣点列表窗体。兴趣点列表窗体类涉及以下几个技术点：

- ❑ 动态创建用来显示兴趣点列表的组件 ListView。
- ❑ 通过 ListView 的 setOnItemClickListener 事件监听单击列表项事件。
- ❑ 通过工具类 WAnalysisFile 中提供的方法，读取当前分段路书的兴趣点集合。

13.16.1　兴趣点列表窗体类框架设计

兴趣点列表窗体的实现大致分下面 3 步来实现：

（1）首先通过 WAnalysisFile 类中提供的 getBigPoiPointList ()方法，获取兴趣点信息列表，并将其保存到兴趣点集合 poiLists 变量中。

（2）然后创建 SimpleAdapter 适配器，数据源是 poiLists 集合中的数据信息。

（3）最后创建 ListView 组件，为该组件添加 SimpleAdapter 适配器，以便显示数据。

Java 代码如下：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
/**
 * 兴趣点列表窗体
 * */
public class TripPoiList extends Activity {
    private LinearLayout poiListLayout;    //用来显示兴趣点的 LinearLayout
    private ListView myListView;           //用来显示兴趣点列表的 ListView
    private TextView tripName;             //用来显示路书名称的 TextView
    private List<PoiPoint> poiLists;       //用来保存兴趣点列表的集合
    private String tripId="";              //用来保存路书 id
    /**
     * 重写 Activity 中的 onCreate 的方法
     * 该方法是在 Activity 创建时被系统调用，是一个 Activity 生命周期的开始
     * @param savedInstanceState：保存 Activity 的状态的
     *      Bundle 类型的数据与 Map 类型的数据相似，都是以 key-value 的形式存储数据的
     * @return
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setTitle(R.string.title);                  //设置标题
        this.setContentView(R.layout.trippoilist); //加载布局资源
        setTripPoiList();                               //显示兴趣点列表
    }
    /**
```

```
     * 兴趣点列表展示
     * @param
     * @return
     */
    public void setTripPoiList(){
        //兴趣点列表上的基本信息
        Bundle bundle = getIntent().getExtras();    //获取 Bundle
        tripId=bundle.getString("tripId");          //获取路书 id 参数
        final String TripName=bundle.getString("tripName");
                                                    //获取路书名字参数
        tripName=(TextView)this.findViewById(R.id.tripName);
                                               //获取路书名字 TextView 组件
        tripName.setText(TripName);                 //显示路书名字

        poiListLayout = (LinearLayout) this.findViewById(R.id.
        poiListLayout);                        //获取兴趣点列表的 LinearLayout

        myListView = new ListView(this);            //创建 ListView 对象
        //创建布局参数对象
        LinearLayout.LayoutParams param3 = new LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.FILL_PARENT,
                LinearLayout.LayoutParams.FILL_PARENT);
        //设置 myListView 高亮显示的颜色，默认是黑色，这里设置为白色
        myListView.setCacheColorHint(Color.WHITE);
        //将 myListView 添加到 poiListLayout 布局上，用 param3 布局参数
        poiListLayout.addView(myListView, param3);
        //创建适配器
        SimpleAdapter adapter = new SimpleAdapter(this, getData(),
                R.layout.trippoilistrow, new String[] { "title", "tel",
                "img",
                         }, new int[] { R.id.poiTitle, R.id.poiTel,
                        R.id.poiImg });
        myListView.setAdapter(adapter);         //为 myListView 添加适配器
        //setViewBinder()方法设置 binder 用于绑定数据到视图，参数为用于绑定数据到
          视图的 binder
        adapter.setViewBinder(new ViewBinder() {

            @Override
            public boolean setViewValue(View arg0, Object arg1,
                    String textRepresentation) {
                //TODO Auto-generated method stub
                if ((arg0 instanceof ImageView) & (arg1 instanceof Bitmap))
{
                    ImageView imageView = (ImageView) arg0;
                    Bitmap bitmap = (Bitmap) arg1;
                    imageView.setImageBitmap(bitmap);
                    return true;
                } else {
                    return false;
                }
            }
        });
        //myListView 列表项单击事件
        myListView.setOnItemClickListener(new OnItemClickListener(){
            //position 指在 ListView 里的位置，id 是 view 的资源 id
            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1, int
            position,
```

```
                    long id) {
                //TODO Auto-generated method stub
                Intent it=new Intent();                //实例化 Intent
                Bundle poiMsg=new Bundle();            //实例化 Bundle
                it.setClass(TripPoiList.this, PoiDetail.class);
                                                       //设置 Class
                //将当前 ListView 被单击的项目的对象放到 Bundle 中
                poiMsg.putSerializable("poiObj", (Serializable) poiLists.
                get(position));
                poiMsg.putString("tripId", tripId);//将路书 id 放到 Bundle 中
                poiMsg.putString("tripName", TripName);
                                                 //将路书名称放到 Bundle 中
                it.putExtras(poiMsg);            //将该对象作为参数传递给下一个窗体
                startActivity(it);               //启动 Activity
            }
        });
    }
    /**
     * 获取兴趣点列表
     * @return
     */
    private List<Map<String, Object>> getData() {
        ……//此处省略了方法功能实现，将在之后进行介绍
    }
}
```

13.16.2　兴趣点列表 ListView 数据填充

兴趣点列表展示在 ListView 组件上，兴趣点数据保存在 poiList 集合中，通过 getData()方法将兴趣点的名称、图片、地址、电话以（key，value）的形式保存到集合中，作为 SimpleAdapter 的数据源。具体实现代码如下：

```
private List<Map<String, Object>> getData() {
        List<Map<String, Object>> list = new ArrayList<Map<String,
        Object>>();                                  //创建 List 对象
        WAnalysisFile readFile=new WAnalysisFile();
                                                     //创建自定义类 WAnalysisFile 对象
        List<PoiPoint> poiList=readFile.getBigPoiPointList(this,tripId);
                                                     //获取兴趣点列表
        poiLists=poiList;
        for (int i = 0; i < poiList.size(); i += 1) {  //循环获取兴趣点信息
            Map<String, Object> map = new HashMap<String, Object>();
            PoiPoint poi=poiList.get(i);
            map.put("title", poi.getName());    //兴趣点标题存储到 map 中
            String[] poiTel=poi.getTel().split(" ");
                                                //兴趣点多个电话用空格分开的
            String poiTels="";
            for(int j=0;j<poiTel.length;j++){
                if(poiTel.length>2){
                    break;
                }
                poiTels=poiTels+poiTel[j]+", "; //拼接兴趣点电话字符串
            }
            if(!"".equals(poiTels)){
                poiTels=poiTels.substring(0, poiTels.length()-1);
```

```
                                                      //去掉末尾的逗号
            }
            map.put("tel", poiTels);                  //兴趣点电话存储到 map 中

            List<String> poiImgList=poi.getImgList();  //获取图片列表
            if(poiImgList.size()!=0){                 //判断是否有图片
                String tripImg=poi.getImgList().get(0).split("[.]")[0]+".
                jpg";
                InputStream iso = null;
                try {
                    //加载图片
                    iso = this.getAssets().open(tripId+"/SmallRoute"+"/"
                    +poi.getRouteId()+"/images/"+tripImg);
                } catch (IOException e) {
                    //TODO Auto-generated catch block
                    e.printStackTrace();
                }
                Bitmap bitmap = null;
                bitmap = BitmapFactory.decodeStream(iso);
                map.put("img",bitmap);
            }else{
                int picnum=0;
                map.put("img",R.drawable.nocolor);
            }
            list.add(map);
        }
        return list;
    }
```

13.17　兴趣点详情窗体类的设计及实现

在兴趣点列表中单击某一个兴趣点，进入兴趣点详情信息，在详情窗体中可以播放该兴趣介绍的 MP3，可以给该兴趣点场所致电，可以展示从当前位置到该兴趣点的线路走向。

13.17.1　兴趣点详情窗体类的框架设计

兴趣点详情窗体类实现较为复杂，涉及以下几方面：

- 从布局上来看，“致电”、“带我去”、“播放”按钮均是动态添加，如果该兴趣点没有电话信息，则不添加“致电”按钮，如果该兴趣点没有 MP3 信息，则不添加“播放”按钮。
- 本窗体有“带我去”功能，单击该按钮，实现的功能是获取当前兴趣点到当前用户所在位置的线路，所以这里需要开启 GPS 功能。

Java 代码如下：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
/**
 * 兴趣点详情窗体
 * */
public class PoiDetail extends Activity{
```

```
private TextView detailPoiName; //声明 TextView 变量，用来显示兴趣点名称
private TextView detailPoiAddr; //声明 TextView 变量，用来显示兴趣点地址信息
private TextView detailPoiTel;  //声明 TextView 变量，用来显示兴趣点电话信息
private TextView detailPoiDesc; //声明 TextView 变量，用来显示兴趣点描述信息
private ImageView detailPoiPic; //声明 ImageView 变量，用来显示兴趣点图片信息
private LinearLayout detailPicLayout;
                //声明 LinearLayout 变量，用来显示兴趣点图片的布局
private LinearLayout layoutPoiButs;
                //声明 LinearLayout 变量，用来显示兴趣点上按钮的布局
private LinearLayout addrAndTelLayout;
                //声明 LinearLayout 变量，用来显示兴趣点中地址和电话的布局
private LocationManager locationManager;//声明 LocationManager 变量
private PoiPoint poi;                              //声明 PoiPoint 变量
private MediaPlayer player = new MediaPlayer(); //声明 MediaPlayer 变量
/**
 * 重写 Activity 中的 onCreate 的方法
 * 该方法是在 Activity 创建时被系统调用，是一个 Activity 生命周期的开始
 * @param savedInstanceState: 保存 Activity 的状态的
 *     Bundle 类型的数据与 Map 类型的数据相似，都是以 key-value 的形式存储数据的
 * @return
 */
@Override
protected void onCreate(Bundle savedInstanceState) {
    //TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    this.setTitle(R.string.title);                     //设置标题
    this.setContentView(R.layout.poidetail);           //加载布局资源
    setTripDetail();                                   //获取兴趣点详情
}

/**
 * 获取兴趣点详情
 */
public void setTripDetail(){
    Bundle poiMsg=getIntent().getExtras();
    poi=(PoiPoint) poiMsg.getSerializable("poiObj");//获取 PoiPoint 对象
    final String tripId=poiMsg.getString("tripId");//获取路书 id
    String TripName=poiMsg.getString("tripName");  //获取路书名称
    detailPoiName=(TextView)this.findViewById(R.id.detailPoiName);
                                        //获取兴趣点名称 TextView 组件
    detailPoiName.setText(poi.getName()); //设置兴趣点名称
    addrAndTelLayout=(LinearLayout)
    this.findViewById(R.id.addrAndTelLayout);  //兴趣点电话，布局

    String addr=poi.getAddress();           //获取兴趣点的地址
    String tel=poi.getTel();                //获取兴趣点电话
    if(!addr.equals("")){                   //地址非空，显示地址信息
        detailPoiAddr=new TextView(this);   //创建 TextView 对象
        //创建 LayoutParams 对象
        LinearLayout.LayoutParams detailPoiAddrParam = new
        LinearLayout.LayoutParams(LayoutParams.WRAP_CONTENT,
        LayoutParams.WRAP_CONTENT);
        detailPoiAddr.setTextColor(Color.BLACK);//设置兴趣点地址文字颜色
        detailPoiAddr.setText("地址："+addr);   //显示兴趣点地址
        addrAndTelLayout.addView(detailPoiAddr);
```

```
                        //将兴趣点地址添加到 addrAndTelLayout 布局上
        }
        if(!tel.equals("")){
            detailPoiTel=new TextView(this);        //创建 TextView 对象
            //创建 LayoutParams 对象
            LinearLayout.LayoutParams detailPoiTelParam = new
            LinearLayout.LayoutParams(LayoutParams.WRAP CONTENT,
            LayoutParams.WRAP CONTENT);
            detailPoiTel.setTextColor(Color.BLACK); //设置兴趣点电话文字颜色
            detailPoiTel.setText("电话: "+poi.getTel().replace(" ", ","));
                                            //将电话中的空格替换为逗号
            addrAndTelLayout.addView(detailPoiTel);
                        //将兴趣点电话添加到 addrAndTelLayout 布局上
        }
        detailPoiDesc=(TextView)this.findViewById(R.id.detailPoiDesc);
                        //获取兴趣点描述的 TextView 组件
        detailPoiDesc.setText(poi.getDesc().replaceAll("</br>", "\n\n"));
                        //将兴趣点描述中的</br>替换为换行符
        //获取兴趣点图片所在的 LinearLayout 组件
        detailPicLayout=(LinearLayout) this.findViewById(R.id.
        poiDetailPicLayout);

        detailPoiPic=new ImageView(this);       //创建兴趣点图片 ImageView
        //创建布局参数
        LinearLayout.LayoutParams detailPoiPicLayoutParam = new Linear-
        Layout.LayoutParams(LayoutParams.WRAP CONTENT,LayoutParams.
        WRAP CONTENT);
         //为 detailPoiPic 设置布局参数 detailPoiPicLayoutParam
        detailPoiPic.setLayoutParams(detailPoiPicLayoutParam);
        if(poi.getImgList().size()!=0){
            String poiImg=poi.getImgList().get(0).split("[.]")[0]+"_m.
            jpg";                                   //获取图片名
            InputStream iso;
            try {
                //打开 assets 下的图片
                iso = this.getAssets().open(tripId+"/SmallRoute"+"/"+
                poi.getRouteId()+"/images/"+poiImg);
                Bitmap bitmap = null;
                bitmap = BitmapFactory.decodeStream(iso);
                detailPoiPic.setImageBitmap(bitmap);
                detailPicLayout.addView(detailPoiPic);
                                        //将图片添加到 detailPicLayout 上

                detailPicLayout.setOnClickListener(new OnClickListener(){
                                        //图片的单击事件

                    @Override
                    public void onClick(View v) {
                        //TODO Auto-generated method stub
                        Intent it=new Intent();
                        Bundle tripDetailPic=new Bundle();
                        it.setClass(PoiDetail.this, TripDetailPic.class);
                        String tripImg=tripId+"/SmallRoute"+"/"+poi.
                        getRouteId()+"/images/"+"y "+poi. getImgList().
                        get(0).split("[.]")[0]+".jpg";
                        tripDetailPic.putString("tripImgPath",tripImg);
                        it.putExtras(tripDetailPic);
                    startActivity(it);
                    }
```

```
            });
        } catch (IOException e1) {
            //TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }
    //获取当前位置
    locationManager = (LocationManager) getSystemService(Context.
    LOCATION_SERVICE);
    locationManager.requestLocationUpdates(LocationManager.
    GPS_PROVIDER, 1000, 0.0001f, locationListener);
                            //实例化 locationManager 对象并添加监听器
    //获取兴趣点详情页上按钮的 LinearLayout
    layoutPoiButs=(LinearLayout) this.findViewById(R.id.poibuts);
    //定义带我去、致电、播放按钮数组
    int []pic={R.drawable.daiwoqu,R.drawable.zhidian,R.drawable.
    bofang};
    List<ImageButton> imgButtLists=new ArrayList();//创建 List 对象
    for(int i=0;i<3;i+=1){                          //动态摆放 3 个按钮
        final ImageButton imgBut=new ImageButton(this);
                                             //创建 ImageButton 按钮
        if(i==0){
            imgBut.setOnClickListener(new Button.OnClickListener(){
                        //带我去按钮的单击事件
                public void onClick(View arg0) {
                        ……//此处省略了方法功能实现，将在之后进行介绍
                }
            });
            imgBut.setBackgroundResource(pic[i]);
            imgButtLists.add(imgBut);
        }
        if(i==1 && !"".equals(poi.getTel())){
            imgBut.setOnClickListener(new Button.OnClickListener(){
                        //致电按钮的单击事件
                public void onClick(View arg0) {
                    prePoiTel(poi.getTel());
                        //该方法的功能实现，将在之后进行介绍
                }
            });
            imgBut.setBackgroundResource(pic[i]);
                        //设置致电按钮的背景图片
            imgButtLists.add(imgBut);
        }
        if(i==2 && !"".equals(poi.getMp3Path())){
            imgBut.setOnClickListener(new Button.OnClickListener(){
                        //播放按钮的单击事件
                public void onClick(View arg0) {
                        ……//此处省略了方法功能实现，将在之后进行介绍
                }
            });
            imgBut.setBackgroundResource(pic[i]);
            imgButtLists.add(imgBut);
        }
    }

    //统一设置按钮间距
    for(int i=0;i<imgButtLists.size();i++){

        if(imgButtLists.size()==4){
```

```
                LinearLayout.LayoutParams margin = new LinearLayout.
                LayoutParams(LayoutParams.WRAP CONTENT,LayoutParams.
                WRAP CONTENT);
                margin.setMargins(0, 0, 5, 0);
                imgButtLists.get(i).setLayoutParams(margin);
                layoutPoiButs.addView(imgButtLists.get(i));
            }
            if(imgButtLists.size()==3){
                LinearLayout.LayoutParams margin = new LinearLayout.
                LayoutParams(LayoutParams.WRAP CONTENT,LayoutParams.
                WRAP CONTENT);
                margin.setMargins(10, 0,20, 0);
                imgButtLists.get(i).setLayoutParams(margin);
                layoutPoiButs.addView(imgButtLists.get(i));
            }
            if(imgButtLists.size()==2){
                LinearLayout.LayoutParams margin = new LinearLayout.
                LayoutParams(LayoutParams.WRAP_CONTENT,LayoutParams.
                WRAP_CONTENT);
                margin.setMargins(10, 0,40, 0);
                imgButtLists.get(i).setLayoutParams(margin);
                layoutPoiButs.addView(imgButtLists.get(i));
            }
        }
    }
    /**
     * 撤销关闭 MP3
     */
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if(keyCode==4 && player.isPlaying()){
            player.reset();
        }
        return super.onKeyDown(keyCode, event);
    }
    //位置监听器
   private final LocationListener locationListener = new LocationListener()
{

        public void onLocationChanged(Location location) {}

        public void onProviderDisabled(String provider) {}

        public void onProviderEnabled(String provider) {}

        public void onStatusChanged(String provider, int status, Bundle
        extras) {}
    };

    /**
     * 关闭位置监听
     */
    protected void onStop() {
        locationManager.removeUpdates(locationListener);   //移除位置监听器
        super.onStop();
    }

    /**
     * 打开位置监听
     */
    protected void onRestart() {
```

```
        //重新注册并实例化位置监听器
        locationManager = (LocationManager) getSystemService(Context.
        LOCATION_SERVICE);

    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
    1000, 0.0001f, locationListener);
        super.onRestart();
    }
}
```

运行效果如图 13.24 所示。



图 13.24　兴趣点详情展示

13.17.2　带我去功能的实现

单击“带我去”按钮，首先设置 Criteria 参数，通过 getLastKnownLocation()方法获取用户当前位置，然后调用系统的自带的“带我去”功能。具体代码如下：

```
imgBut.setOnClickListener(new Button.OnClickListener() {
                                                    //“带我去”按钮的单击事件
                    public void onClick(View arg0) {
                        Criteria criteria = new Criteria();
                        criteria.setAccuracy(Criteria.ACCURACY_FINE);
                        criteria.setAltitudeRequired(false);
                        criteria.setBearingRequired(false);
                        criteria.setCostAllowed(false);
                        criteria.setPowerRequirement(Criteria.POWER_LOW);
                        //获取当前已知的最后一个地理位置
                        Location location = locationManager.getLastKnown-
                        Location(locationManager.getBestProvider(criteria,
                        true));
                        if (location == null) {
                            Toast.makeText(PoiDetail.this, "没有 GPS 信号",
                                    Toast.LENGTH_SHORT).show();
```

```
                    } else {
                        Intent intent = new Intent();
                    intent.setAction(android.content.Intent.ACTION_VIEW);
                    intent.setData(Uri.parse("http://maps.google.com/
                    maps?f=d&saddr="+(location.getLatitude()+","+
                    location.getLongitude())+"&daddr="+(poi.getLat()+","
                    +poi.getLon())+"&hl=cn"));
                    startActivity(intent);
                        }
                    }
                });
```

13.17.3　致电功能的实现

用户单击“致电”按钮时，如果是该兴趣点有多个电话信息，则以弹出框 AlertDialog 的形式显示。单击某一个电话，通过 Intent myIntentDial = new Intent("android.intent.action.DIAL",Uri.parse("tel:"+tel))，调用系统的致电软键盘，并将当前的选择的电话号码显示在致电软键盘中。

```
public void prePoiTel(String tel){
        final String temp[]=tel.split(" ");     //多个电话之间用空格“ ”分开
        if(temp.length>1){                  //如果电话是多于 1 个，以对话框的形式显示
            //显示电话列表的对话框
            new AlertDialog.Builder(PoiDetail.this)
          .setTitle("选择")
          .setItems(temp,
          new DialogInterface.OnClickListener(){
            public void onClick(DialogInterface dialog, int whichcountry){
                                           //电话列表对话框单击事件
              telToPoi(temp[whichcountry]);//调用致电的自定义方法
            }
          })
         .setNegativeButton("取消", new DialogInterface.OnClickListener(){
            public void onClick(DialogInterface d, int which){
              d.dismiss();                 //关闭对话框
            }
          })
          .show();
        }
        else{
            telToPoi(temp[0]);
        }
    }

    /**
     * 致电兴趣点
     * @param tel: 电话
     */
    public void telToPoi(String tel){
        Intent myIntentDial = new Intent("android.intent.action.DIAL",
        Uri.parse("tel:"+tel));
        startActivity(myIntentDial);
    }
```

13.17.4　播放 MP3 功能的实现

单击“播放”按钮，通过 MediaPlayer 组件播放 MP3，同时“播放”按钮变为“停止”。具体代码实现如下：

```
imgBut.setOnClickListener(new Button.OnClickListener() {
                                        //“播放”按钮的单击事件
                public void onClick(View arg0) {
                    try{
                        if(player.isPlaying()){
                            player.reset();
                            imgBut.setBackgroundResource(R.drawable.
                            bofang);
                        }
                        else{
                            player.reset();

                            AssetFileDescriptor afd = PoiDetail.this.
                            getAssets().openFd(tripId+"/SmallRoute
                            "+"/"+poi.getRouteId()+"/mp3/"+poi.
                            getMp3Path().toLowerCase());
                            player.setDataSource(afd.getFileDescrip-
                            tor(),afd.getStartOffset(),afd.
                            getLength());
                            player.prepare();
                            player.start();
                            imgBut.setBackgroundResource(R.drawable.
                            stop);
                        }
                        player.setOnCompletionListener(new OnComple-
                        tionListener(){
                            public void onCompletion(MediaPlayer mp) {
                                imgBut.setBackgroundResource(R.
                                drawable.bofang);
                            }
                        });

                    }
                    catch(Exception e){
                        e.printStackTrace();
                    }
                }
            });
```

13.18　服务区列表窗体类的设计及实现

单击图 13.4 中的“服务区”按钮，出现服务区列表信息。该窗体的实现涉及以下几个技术点：

- 动态创建用来显示服务区列表的组件 ListView。
- 通过 ListView 的 setOnItemClickListener 事件监听单击列表项事件。

13.18.1　服务区列表窗体类的框架设计

服务区列表的设计思路及实现流程如下：

（1）首先通过 WAnalysisFile 类中提供的 searchBeetlsByKeyword()方法，获取服务区信息列表，并将其保存到商品集合 goodsList 变量中。

（2）然后创建 SimpleAdapter 适配器，数据源是 beetleList 集合中的数据信息。

（3）最后创建 ListView 组件，为该组件添加 SimpleAdapter 适配器，以便显示数据。

Java 代码如下：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
/**
 * 服务区列表窗体
 * */
public class DZDealersList extends Activity {
    private LinearLayout dealerListLayout; //用来显示服务区的列表的布局
    private ListView myListView;        //用来显示服务区列表的 ListVIew
    private List<Beetle> beetleList = new ArrayList();
                                        //声明 List 变量，保存获取的服务区信息
    private ImageButton searchButt;     //搜索按钮的 ImageButton
    private TextView searchPro;         //声明 TextView 变量，显示“省”提示信息
    private TextView searchCity;        //声明 TextView 变量
    private ProgressDialog myDialog;    //声明 ProgressDialog 变量
    private TextView dealerPro;         //显示服务区省份的 TextView
    private String pro = "";            //声明 String 变量
    private String city = "";           //声明 String 变量

    /**
     * 重写 Activity 中的 onCreate 的方法。 该方法是在 Activity 创建时被系统调用，是
     一个 Activity 生命周期的开始
     *
     * @param savedInstanceState
     *           ：保存 Activity 的状态的。 Bundle 类型的数据与 Map 类型的数据相似，
     都是以 key-value 的形式存储数据的
     * @return
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setTitle(R.string.title);                  //设置标题
        this.setContentView(R.layout.dzdealerslist);    //加载布局资源
        setDealersList();
    }
    /**
     * 获取服务区列表
     */
    public void setDealersList() {
        //动态生成服务器列表
        dealerListLayout = (LinearLayout) this
                .findViewById(R.id.dealerListLayout);
                                        //获取服务区列表的 LinearLayout
```

```
        myListView = new ListView(this);                //创建 ListView 对象
        //创建布局参数
        LinearLayout.LayoutParams param3 = new LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.FILL_PARENT,
                LinearLayout.LayoutParams.FILL_PARENT);
        myListView.setCacheColorHint(Color.WHITE);
                                    //设置 myListView 高亮显示颜色，默认黑色
        //为 dealerListLayout 添加 myListView 组件，布局参数为 param3
        dealerListLayout.addView(myListView, param3);
        setDealersAdapter();        //设置 myListView 的 adapter()方法
        //myListView 的下拉项单击事件
        myListView.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1, int
            arg2,
                    long arg3) {
                //TODO Auto-generated method stub
                Intent it = new Intent();           //实例化 Intent
                Bundle beanMsg = new Bundle();      //实例化 Bundle
                it.setClass(DZDealersList.this, DZDealersDetail.class);
                                                    //设置 Class
                beanMsg.putSerializable("beanObj", (Serializable) beetleList
                        .get(arg2));        //将服务器对象保存到 beanMsg 中
                it.putExtras(beanMsg);      //将该对象作为参数传递给下一个窗体
                startActivity(it);          //启动 Activity
            }
        });
        dealerPro = (TextView) this.findViewById(R.id.dealerPro);
                                            //获取服务器的省份 TextView 组件
        dealerPro.setOnClickListener(new OnClickListener() {
                                            //省份单击事件
                    public void onClick(View v) {
                        prePro();
                    }
                });
        searchPro = (TextView) this.findViewById(R.id.dealerPro);
                                            //获取省市 TextView 组件

        searchButt = (ImageButton) this.findViewById(R.id.dealerSearch);
                                            //搜索按钮 ImageButton 组件
        searchButt.setOnClickListener(new OnClickListener() {
                                            //搜索按钮单击事件

                    @Override
                    public void onClick(View v) {
                        //TODO Auto-generated method stub
                        get4S();
                    }

                });
        get4S();
    }

    /**
     * 请求数据，获取服务器信息
     */
    public void get4S() {
```

```
        ……//此处省略了方法功能实现，将在之后进行介绍
    }
    /**
     * 设置 adapter
     */
    private void setDealersAdapter() {
        ……//此处省略了方法功能实现，将在之后进行介绍
    }

    /**
     * 获取服务器列表数据
     *
     * @return
     */
    private List<Map<String, Object>> getData() {
        ……//此处省略了方法功能实现，将在之后进行介绍
    }

    /**
     * 省市下拉菜单
     */
    public void prePro() {
        final String temp[] = StaticString.pro;
        //用来显示省市的对话框
        new AlertDialog.Builder(DZDealersList.this).setTitle("选择").
        setItems(
                temp, new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int
                    whichcountry) {
                        //将选中的省份信息显示在 TextView 上
                        dealerPro.setText("  " + temp[whichcountry]);
                    }
                }).setNegativeButton("取消",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface d, int which) {
                        d.dismiss();   //关闭对话框
                    }
                }).show();
    }
}
```

13.18.2　服务区列表 ListView 数据填充

服务区列表展示在 ListView 组件上，服务区列表数据保存在 beetleList 集合中，通过 getData()方法，将服务区的名称、服务区的电话、服务区的地址以(key,value)的形式保存到集合中，作为 SimpleAdapter 的数据源。具体实现代码如下：

```
public void get4S() {
        myDialog = ProgressDialog.show(DZDealersList.this, "请稍等...",
                "数据检索中...", true);
        new Thread() {
            public void run() {
                try {
                    pro = searchPro.getText().toString().trim();
```

```
                                                    //获取当前要搜索的省份信息
                    //查询该省份的服务区信息列表，保存到 beetleList 集合中
                    beetleList = new WAnalysisFile().searchBeetlsBy-
                    Keyword(
                            DZDealersList.this, "", pro);
                    Message m = new Message();
                    poiHandler.sendMessage(m);
                } catch (Exception e) {
                    e.printStackTrace();
                } finally {
                    myDialog.dismiss();
                }
            }
        }.start();
    }

    //兴趣点详情 Handler
    Handler poiHandler = new Handler() {
        public void handleMessage(Message msg) {
            setDealersAdapter();
        }
    };

    /**
     * 设置 adapter
     */
    private void setDealersAdapter() {
        OverrideAdapter adapter = new OverrideAdapter(DZDealersList.this,
                getData(), R.layout.dzdealerslistrow, new String[] {
                        "dealerName", "dealerAddr", "dealerTel", }, new
                        int[] {
                        R.id.dealersName, R.id.dealersAddr,
                        R.id.dealersTel });
        myListView.setAdapter(adapter);
    }

    /**
     * 获取服务器列表数据
     *
     * @return
     */
    private List<Map<String, Object>> getData() {
        List<Map<String, Object>> list = new ArrayList<Map<String,
        Object>>();
        for (int i = 0; i < beetleList.size(); i += 1) {
            Map<String, Object> map = new HashMap<String, Object>();
            Beetle s4 = beetleList.get(i);
            map.put("dealerName", s4.getName());
            map.put("dealerAddr", s4.getAddress());
            map.put("dealerTel", s4.getTel().replace("  ", ","));
            list.add(map);
        }
        return list;
    }
```

运行效果如图 13.25 所示。

单击省文本框，可以选择省份进行搜索，如图 13.26 所示。

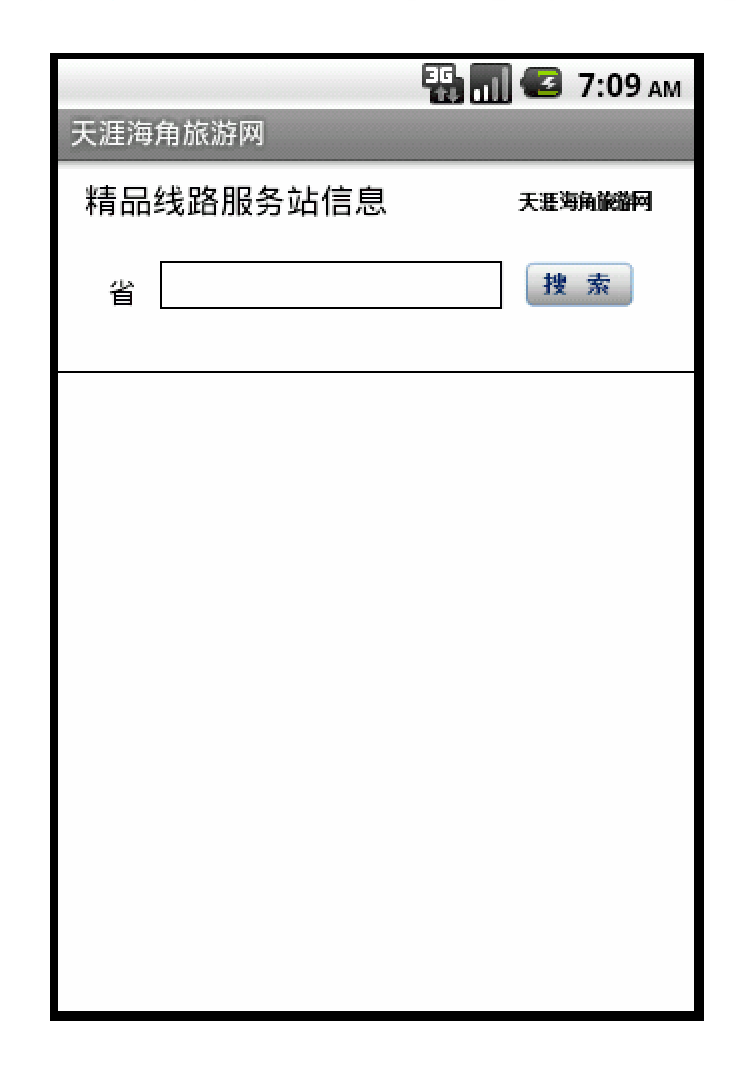



图 13.25　服务区列表



图 13.26　选择省份

13.19　服务区详情窗体类的设计及实现

单击服务区列表项，进入服务区详情展示窗体。

Java 代码如下：

```
package com.tyhj;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class DZDealersDetail extends Activity {
    private TextView detailDealerName;      //用来显示服务区名称的 TextView
    private TextView detailDealerAddr;      //用来显示服务区地址的 TextView
    private TextView detailDealerPerson;    //用来显示服务区联系人的 TextView
    private TextView detailDealerTel;       //用来显示服务区电话的 TextView
    private Beetle s4;                      //声明服务区变量

    private LocationManager locationManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setTitle(R.string.title);      //设置标题栏上显示的文字
        this.setContentView(R.layout.dzdealersdetail);
                                            //加载窗体布局资源文件
        setDealersDetail();         //调用自定义方法，将数据显示到窗体对应组件上
    }

    public void setDealersDetail() {

        Bundle beanMsg = getIntent().getExtras();
                                            //获取窗体传过来的 Bundle 对象
        s4 = (Beetle) beanMsg.getSerializable("beanObj");
                                            //从 Bundle 中获取服务区对象
```

```
detailDealerName = (TextView) this.findViewById(R.id.
detailDealerName);
detailDealerName.setText(s4.getName()); //将服务区名称显示在组件上

detailDealerAddr = (TextView) this.findViewById(R.id.
detailDealerAddr);
detailDealerAddr.setText(s4.getAddress());//将服务区地址显示在组件上

detailDealerPerson = (TextView) this
        .findViewById(R.id.detailDealerPerson);
detailDealerPerson.setText(s4.getContacts());
                                       //将服务器的联系人显示在组件上

detailDealerTel = (TextView) this.findViewById(R.id.
detailDealerTel);
detailDealerTel.setText(s4.getTel().replace("  ", ", "));
                                       //将服务器电话显示在组件上
//实例化并注册位置监听器
locationManager = (LocationManager) getSystemService(Context.
LOCATION_SERVICE);
locationManager.requestLocationUpdates(LocationManager.
GPS_PROVIDER, 1000, 0.0001f, locationListener);

ImageButton go = (ImageButton) this
        .findViewById(R.id.detailDealerTakeme);
go.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View arg0) {
        Criteria criteria = new Criteria();
        criteria.setAccuracy(Criteria.ACCURACY_FINE);
        criteria.setAltitudeRequired(false);
        criteria.setBearingRequired(false);
        criteria.setCostAllowed(false);
        criteria.setPowerRequirement(Criteria.POWER_LOW);
        //获取当前已知的最后一个地理位置
        Location location = locationManager
                .getLastKnownLocation(locationManager.
                getBestProvider(
                        criteria, true));
        if (location == null) {
            Toast.makeText(DZDealersDetail.this, "没有 GPS 信号",
                    Toast.LENGTH_SHORT).show();
        } else {
            Intent intent = new Intent();
            intent.setAction(android.content.Intent.ACTION_VIEW);
            intent.setData(Uri
                    .parse("http://maps.google.com/maps?f=d&saddr="
                            + (location.getLatitude() + "," +
                            location
                                    .getLongitude()) + "&daddr="
                            + (s4.getLat() + "," + s4.getLon())
                            + "&hl=cn"));
            startActivity(intent);
        }
    }
});
//获取资源文件中的 ImageButton
ImageButton tel = (ImageButton) this.findViewById(R.id.
detailDealerZD);
//致电服务区按钮的单击事件
```

```
        tel.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View arg0) {
                prePoiTel(s4.getTel());
            }
        });
        ImageButton map = (ImageButton) this.findViewById(R.id.
        detailDealerMap);
        map.setOnClickListener(new Button.OnClickListener() {
                                                    //单击地图按钮，跳到地图页面
            public void onClick(View arg0) {
                Bundle bl = new Bundle();
                bl.putString("id", s4.getId());
                bl.putString("name", s4.getName());
                bl.putString("loc", s4.getLat() + " " + s4.getLon());
                bl.putBoolean("pan", false);
                Intent it = new Intent();
                it.setClass(DZDealersDetail.this, RoadMapView.class);
                it.putExtras(bl);
                startActivity(it);
            }
        });

    }

    //位置监听器
    private final LocationListener locationListener = new LocationListener() {

        public void onLocationChanged(Location location) {
        }

        public void onProviderDisabled(String provider) {
        }

        public void onProviderEnabled(String provider) {
        }

        public void onStatusChanged(String provider, int status, Bundle
        extras) {
        }
    };

    //关闭位置监听
    protected void onStop() {
        locationManager.removeUpdates(locationListener);//注销位置监听器
        super.onStop();
    }

    //打开位置监听
    protected void onRestart() {
        //重新注册并实例化位置监听器
        locationManager = (LocationManager) getSystemService(Context.
        LOCATION SERVICE);
        locationManager.requestLocationUpdates(LocationManager.
        GPS PROVIDER, 1000, 0.0001f, locationListener);
        super.onRestart();
    }

    //准备电话
    public void prePoiTel(String tel) {
        final String temp[] = tel.split("  ");
```

```
        if (temp.length > 1) {
            new AlertDialog.Builder(DZDealersDetail.this).setTitle("选择")
                    .setItems(temp, new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog,
                                int whichcountry) {
                            telToPoi(temp[whichcountry]);
                        }
                    }).setNegativeButton("取消",
                            new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface d, int
                                which) {
                                    d.dismiss();
                                }
                            }).show();
        } else {
            telToPoi(temp[0]);
        }
    }

    //致电兴趣点
    public void telToPoi(String tel) {
        Intent myIntentDial = new Intent("android.intent.action.DIAL", Uri
                .parse("tel:" + tel));
        startActivity(myIntentDial);
    }
}
```

13.20 项目技术难点

到本节为止，本项目基本功能已经介绍完毕，项目在实现过程中的技术难点大概有以下几方面。

- 在手机应用中，用户同地图的交互是通过触摸屏幕的方式完成的。因此，掌握这套接口中，负责屏幕坐标同地理坐标相互转换的类和方法，是实现一切互动操作需求的关键。
- 在众多的需求中，获取地图上某一图符被单击的事件是最重要的交互手段。而地图标注是以地图附加层的形式出现在接口中的。掌握接口中的地图附加层，尤其是 OverlayItem 和 ItemizedOverlay 两个类的使用，是十分关键的。

我们习惯了通过 XML 去布局窗体界面，但有的时候需要动态的通过代码去创建组件，创建布局参数。在本案例中，对动态布局应用的淋漓尽致，例如，分段线路列表、兴趣点详情等，这一点也是大家要掌握的技术点。

第 14 章　乐乐网上购物商城——边走边购物

随着上网速度更快的 3G 的推出，手机购物成为现实，并日渐受到大众的欢迎。如果 3G 网络能够实现无缝覆盖，那么势必会有越来越多的用户选择“边走边购物”的手机购物模式，而手机购物也将成为“网络购物”之后，人们购物模式的又一次“升级”。作为 Android 的开发人员，势必需要能够胜任关于手机网络购物应用的设计与实现。本章案例模拟了手机端网络购物的实现，读者通过本案例可以深入了解手机客户端请求服务器资源的实现，以及加载网络图片、常用布局的使用等技术。

14.1　网上商城功能概述

乐乐网上购物商城实现以下基本功能模块：

- ❑ 推荐商品列表展示；
- ❑ 家用电器商品列表展示；
- ❑ 手机数码商品列表展示；
- ❑ 电脑办公商品列表展示；
- ❑ 商品详情展示；
- ❑ 商品添加到购物车功能；
- ❑ 购物车列表展示；
- ❑ 订单提交功能实现；
- ❑ 登录功能实现；
- ❑ 订单列表展示。

本项目分为服务器端和客户端两部分代码。服务器端提供商品信息列表、订单信息列表、用户基本信息。客户端通过请求服务器端数据来展示商品信息。下面介绍一下该项目实现流程：

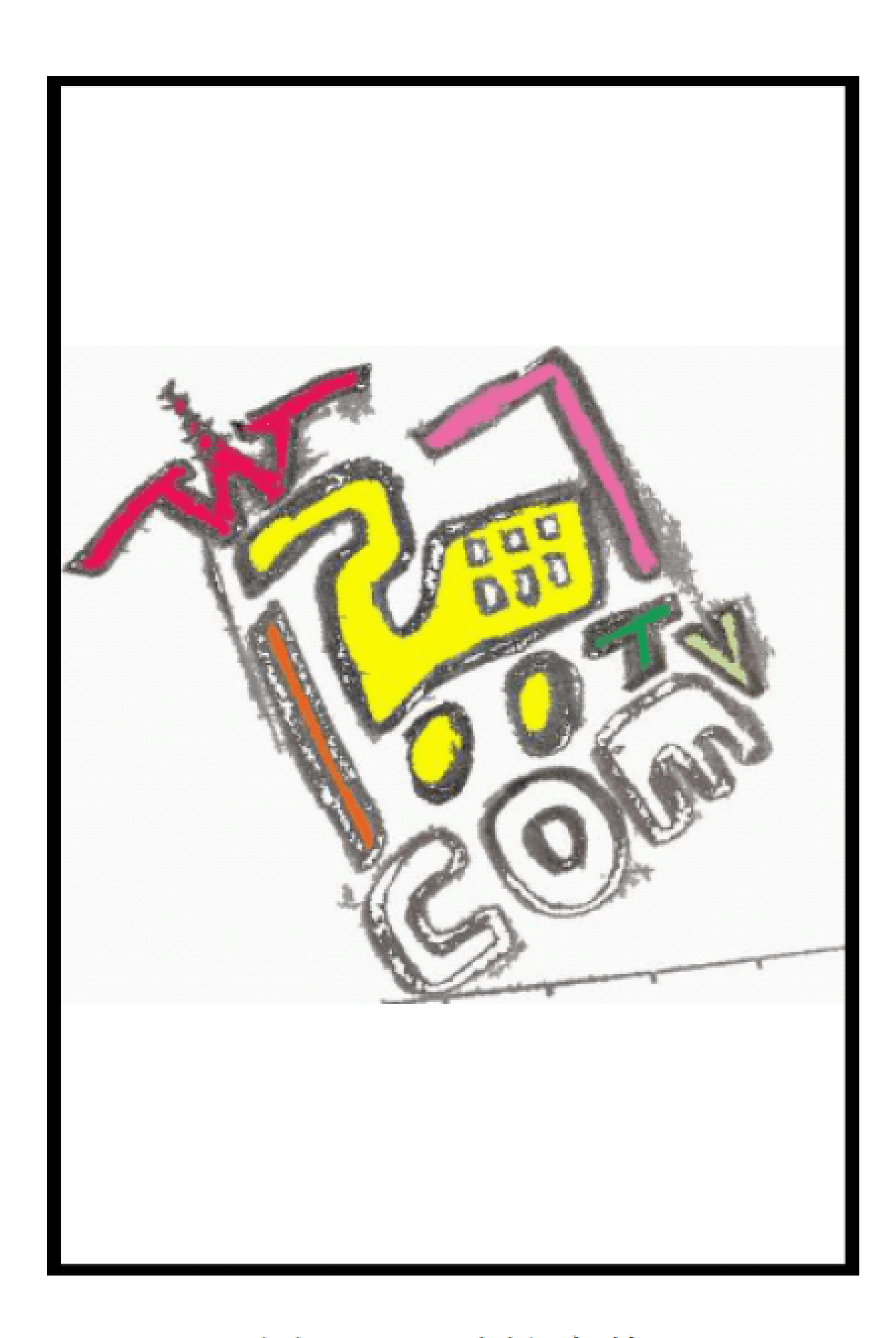


图 14.1　欢迎窗体

（1）启动项目后，首先展示的是乐乐网上购物商城 Logo 画面，如图 14.1 所示。随后进入应用的主界面，手机客户端请求服务器端推荐商品列表来展示商品信息，如图 14.2 所示。

（2）单击主窗体中的“家用电器”按钮，请求服务器资源，展示家用电器信息列表，如图 14.3 所示，同样，单击“手机数码”及“电脑办公”按钮展示相关数据列表，如图

14.4 和图 14.5 所示。



图 14.2　主窗体



图 14.3　家用电器列表



图 14.4　手机数码列表



图 14.5　电脑办公列表

（3）单击列表中的一项商品，进入商品详情信息展示，如图 14.6 所示。单击“购物车”按钮用户可以将该商品放入购物车；单击 menu 按钮，屏幕下方出现查看“购物车”列表菜单，如图 14.7 所示；单击“购物车”按钮，用户可以查询当前购物车上的商品信息，如图 14.8 所示。

（4）单击图 14.8 中的“结算”按钮，进入结算金额流程，首先用户必须登录后才可以

进行结算费用，如果用户未登录该应用，单击“结算”按钮后，跳到用户登录页面，如图 14.9 所示。如果用户登录了该应用，单击“结算”按钮后，直接跳到提交订单窗体，如图 14.10 所示。



图 14.6　商品详情展示窗体



图 14.7　添加购物车



图 14.8　购物车列表窗体



图 14.9　登录窗体图

（5）单击“提交订单”按钮，提交用户的订单信息到服务器，并展示该用户的历史订单列表，如图 14.11 所示。



图 14.10　提交订单信息



图 14.11　订单列表窗体

14.2　系统包、资源规划的准备工作

本项目在实现的过程中，所有客户端用到的图片资源在 res/drawable 下，布局文件在 res/layout 下，字符串常量在 res/values/strings.xml 中，数组常量在 res/values/arrays.xml 中。

14.3　服务器端的开发

在该项目实现过程中，服务器端负责存储用户数据信息、订单信息、商品数据信息等，客户端通过向服务器端发送请求获取相关的数据资源及保存相关的数据资源。

14.3.1　服务器端数据库设计

本案例中商品数据信息、登录时的用户信息验证、订单信息的保存，都是通过请求服务器获取的数据资源。服务器端的数据库为 shop，共有 3 张数据表，即 bill（订单表）、goods（商品表）、users（用户表）。

bill 表记录用户订单信息，结构如下：

```
CREATE TABLE 'bill' (
  'Id' int(11) NOT NULL auto_increment,            //自动编号
  'uid' varchar(20) default '',                    //用户编号
  'gids' varchar(255) default '',                  //订单商品编号
  'gnums' varchar(255) default '',                 //订单商品数量
  'state' varchar(20) default '',                  //订单状态
```

```
  'btime' varchar(10) default '',                    //送货时间
  'btype' varchar(10) default '',                    //付款方式
  'ctime' varchar(20) default '',                    //订单时间
  'address' varchar(255) default '',                 //送货地址
  PRIMARY KEY  ('Id')
) ENGINE=MyISAM AUTO_INCREMENT=16 DEFAULT CHARSET=utf8;
```

goods 表记录商品信息，结构如下：

```
CREATE TABLE 'goods' (
  'Id' int(11) NOT NULL auto_increment,              //自动编号
  'brand' varchar(20) default '',                    //品牌
  'price' float default '0',                         //价格
  'discount' float default '0',                      //折扣
  'bcount' int(11) default '0',                      //购买次数
  'des' text,                                        //描述
  'pic' varchar(20) default '',                      //图片名称
  'dir' varchar(20) default '',                      //图片路径
  'gid' varchar(20) default '',                      //商品编号
  'type' int(11) default '0',                        //商品种类
  'pop' int(11) default '0',                         //是否推荐
  PRIMARY KEY  ('Id')
) ENGINE=MyISAM AUTO_INCREMENT=13 DEFAULT CHARSET=utf8;
```

users 表记录用户信息，结构如下：

```
CREATE TABLE 'users' (
  'Id' int(11) NOT NULL auto_increment,              //自动编号
  'uid' varchar(20) default '',                      //用户名
  'pwd' varchar(20) default '',                      //密码
  PRIMARY KEY  ('Id')
) ENGINE=MyISAM AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

14.3.2　服务器端的简要介绍

本节中对服务器端的相关类进行简单的介绍，以便读者可以充分了解服务器功能的实现，以及相关类的设计。服务器端程序采用 struts2+hiberbate 方式搭建，程序结构如图 14.12 所示。

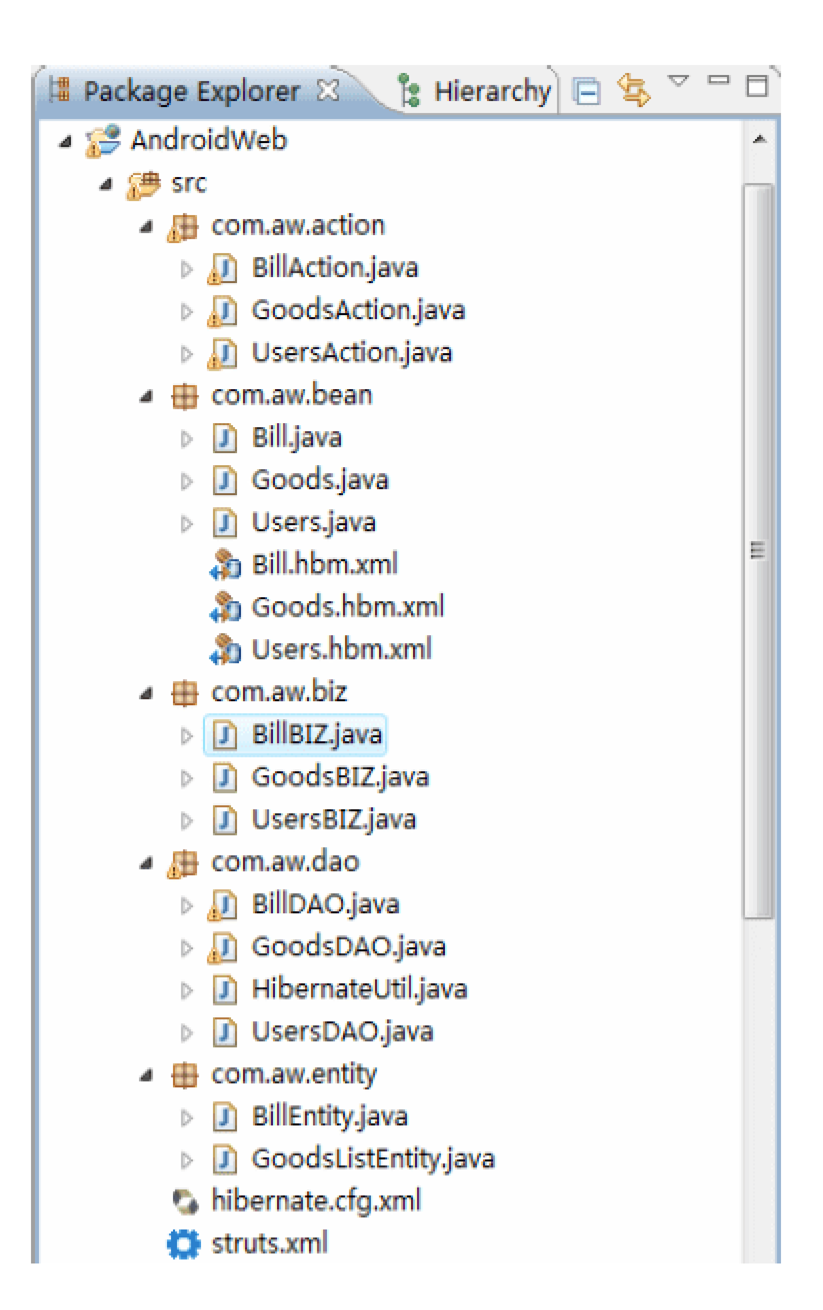


图 14.12　服务器端项目结构

下面介绍一下服务器端相关包的功能及主要的类功能。

- com.aw.action 包：负责响应手机端发出的请求。该包下共有 3 个 action，类及类的功能分别如下。
  - BillAction.java 负责订单类的请求，请求分为列表查询和添加订单两类，结果以 JSON 的形式返回。
  - GoodsAction.java 负责商品类的请求，请求分为推荐商品和分类商品两种查询。
  - UsersAction.java 负责用户的登录请求。

- com.aw.biz 包：负责业务处理。该包下的类及类的功能如下。
  - BillBIZ.java 负责账单业务处理，该类有两个方法，分别是根据用户名获取账单列表的 getBillListByUid()方法和添加新账单的 addBill()方法。
  - GoodsBIZ.java 负责商品的业务处理，该类有两个方法，分别是获取推荐商品列表的 getGoodsPopList()方法和获取分类商品列表的 getGoodsListByType()方法。
  - UsersBIZ.java 负责用户的业务处理，该类有负责用户登录的 userLogin()方法。
- com.aw.dao：包负责数据处理，该包下的类及类的功能如下。
  - BillDAO.java 负责订单的数据操作，该类有两个方法，分别是根据用户名获取账单列表的 getBillListByUid()方法和添加新账单的 addBill()方法。
  - GoodsDAO.java 负责商品的数据处理，该类有负责获取商品列表的方法。
  - UsersDAO.java 负责用户的数据处理，该类有负责用户登录的方法。

14.3.3　服务器端的代码详细介绍

BillAction.java 中的请求分为列表查询和添加订单两类，结果以 JSON 的形式返回。代码如下：

```
package com.aw.action;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class BillAction extends ActionSupport{
    private List<BillEntity> blist;
    public String execute() throws Exception {
        HttpServletResponse response=ServletActionContext.getResponse();
        HttpServletRequest request=ServletActionContext.getRequest();

        response.setCharacterEncoding("utf-8");
        response.setContentType("text/html");

        String type=request.getParameter("type");
        if("list".equals(type)){                          //列表查询
            String uid=request.getParameter("uid");
            this.setBlist(new BillBIZ().getBillListByUid(uid));
        }
        else if("add".equals(type)){                      //订单添加
            String uid=request.getParameter("uid");
            String gids=request.getParameter("gids");
            String gnums=request.getParameter("gnums");
            String btime=request.getParameter("btime");
            String btype=request.getParameter("btype");
            String address=request.getParameter("address");
            boolean pan=new BillBIZ().addBill(uid, gids, gnums, btime,
            btype,address);
            try {
                response.getWriter().println("{\"msg\":\""+pan+"\"}");
            } catch (Exception e) {
                e.printStackTrace();
            }
            return null;
        }

        return SUCCESS;
```

```
    }
    public List<BillEntity> getBlist() {
        return blist;
    }
    public void setBlist(List<BillEntity> blist) {
        this.blist = blist;
    }
}
```

GoodsAction.java 中的请求分为推荐商品和分类商品两种查询，代码如下：

```
package com.aw.action;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class GoodsAction extends ActionSupport{
    List<Goods> glist;

    public String execute() throws Exception {
        HttpServletResponse response=ServletActionContext.getResponse();
        HttpServletRequest request=ServletActionContext.getRequest();

        response.setCharacterEncoding("utf-8");
        response.setContentType("text/html");

        String type=request.getParameter("type");
        if(type.equals("pop")){                              //推荐商品列表
            this.setGlist(new GoodsBIZ().getGoodsPopList());
        }
        else if(type.equals("type")){                        //分类商品列表
            int gtype=Integer.valueOf(request.getParameter("gtype"));
            this.setGlist(new GoodsBIZ().getGoodsListByType(gtype));
        }
        return SUCCESS;
    }

    public List<Goods> getGlist() {
        return glist;
    }

    public void setGlist(List<Goods> glist) {
        this.glist = glist;
    }
}
```

UsersAction.java 中用户的登录请求，代码如下：

```
package com.aw.action;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class UsersAction extends ActionSupport {

    private Users user;

    public String execute() throws Exception {
        HttpServletResponse response=ServletActionContext.getResponse();
        HttpServletRequest request=ServletActionContext.getRequest();

        response.setCharacterEncoding("utf-8");
        response.setContentType("text/html");

        String uid=request.getParameter("uid");
        String pwd=request.getParameter("pwd");
```

```
        this.setUser(new UsersBIZ().userLogin(uid, pwd));

        return SUCCESS;
    }
    public Users getUser() {
        return user;
    }
    public void setUser(Users user) {
        this.user = user;
    }
}
```

BillBIZ.java 类的两个方法，分别是根据用户名获取账单列表的 getBillListByUid()方法和添加新账单的 addBill()方法，代码如下：

```
package com.aw.biz;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class BillBIZ {
    //根据 uid 获取账单列表
    public List<BillEntity> getBillListByUid(String uid){
        return new BillDAO().getBillListByUid(uid);
    }
    //添加账单
    public boolean addBill(String uid,String gids,String gnums,String
    btime,String btype,String address){
        Bill bill=new Bill();
        bill.setUid(uid);
        bill.setGids(gids);
        bill.setGnums(gnums);
        bill.setBtime(changeToWord(btime));
        bill.setBtype(changeToWord(btype));
        bill.setAddress(changeToWord(address));
        SimpleDateFormat s=new SimpleDateFormat("yyyy-MM-dd");
        bill.setCtime(s.format(new Date()));
        bill.setState("waiting");
        return new BillDAO().addBill(bill);
    }
}
```

GoodsBIZ.java 类的两个方法，分别是获取推荐商品列表的 getGoodsPopList()方法和获取分类商品列表的 getGoodsListByType()方法，代码如下：

```
package com.aw.biz;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class GoodsBIZ {

    public List<Goods> getGoodsPopList(){
        String hql="from Goods where pop=1";
        List<Goods> glist=new GoodsDAO().getGoodsByHql(hql);
        return glist;
    }

    public List<Goods> getGoodsListByType(int type){
        String hql="from Goods where type="+type;

        List<Goods> glist=new GoodsDAO().getGoodsByHql(hql);
        return glist;
    }
}
```

UsersBIZ.java 类中的提供负责用户登录的 userLogin()方法，代码如下：

```
package com.aw.biz;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class UsersBIZ {

    public Users userLogin(String uid,String pwd){
        Users user=new UsersDAO().getUserByUid(uid);
        return user;
    }
}
```

BillDAO.java 类有两个方法，分别是根据用户名获取账单列表的 getBillListByUid()方法和添加新账单的 addBill()方法，代码如下：

```
package com.aw.dao;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class BillDAO {

    public List<BillEntity> getBillListByUid(String uid){
        List<BillEntity> blist=new ArrayList<BillEntity>();
        Session session = null;
        try{
            session = HibernateUtil.getSession();
            String hql="from Bill where uid='"+uid+"'";
            Query query = session.createQuery(hql);
            List<Bill> mylist=query.list();
            for(int i=0;i<mylist.size();i+=1){
                BillEntity be=new BillEntity();
                be.setId(mylist.get(i).getId());
                be.setState(mylist.get(i).getState());
                be.setBtime(mylist.get(i).getBtime());
                be.setBtype(mylist.get(i).getBtype());
                be.setCtime(mylist.get(i).getCtime());
                be.setAddress(mylist.get(i).getAddress());
                String gids=mylist.get(i).getGids();
                String gnums[]=mylist.get(i).getGnums().split(",");
                List<GoodsListEntity> glist=new ArrayList<Goods-
                ListEntity>();
                hql="select brand from Goods where id in ("+gids+")";
                List<String> temp=session.createQuery(hql).list();
                for(int j=0;j<temp.size();j+=1){
                    GoodsListEntity ge=new GoodsListEntity();
                    ge.setGname(temp.get(j));
                    ge.setGnum(Integer.valueOf(gnums[j]));
                    glist.add(ge);
                }
                be.setGlist(glist);
                blist.add(be);
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
        finally{
            HibernateUtil.closeSession(session);
        }
        return blist;
    }
```

```
    public boolean addBill(Bill bill){
        boolean pan=true;
        Session session = null;
        try{
            session = HibernateUtil.getSession();
            session.beginTransaction();
            session.saveOrUpdate(bill);
            session.getTransaction().commit();
        }
        catch(Exception e){
            e.printStackTrace();
            session.getTransaction().rollback();
            pan=false;
        }
        finally{
            HibernateUtil.closeSession(session);
        }

        return pan;
    }
}
```

GoodsDAO.java 类中提供负责获取商品列表的方法，代码如下：

```
package com.aw.dao;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class GoodsDAO {

    public List<Goods> getGoodsByHql(String hql){
        List<Goods> glist=new ArrayList<Goods>();
        Session session = null;
        try{
            session = HibernateUtil.getSession();
            Query query = session.createQuery(hql);
            glist=query.list();
        }
        catch(Exception e){
            e.printStackTrace();
        }
        finally{
            HibernateUtil.closeSession(session);
        }
        return glist;
    }
}
```

UsersDAO.java 类中提供用户登录的方法，代码如下：

```
package com.aw.dao;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class UsersDAO {

    public Users getUserByUid(String uid){
        Users user=null;
        Session session = null;
        try{
            session = HibernateUtil.getSession();
            String hql="from Users where uid='"+uid+"'";
            Query query = session.createQuery(hql);
            user=(Users)query.uniqueResult();
        }
        catch(Exception e){
```

```
            e.printStackTrace();
        }
        finally{
            HibernateUtil.closeSession(session);
        }
        return user;
    }
}
```

14.4　手机客户端访问资源权限配置

本项目中涉及读取网络资源，需要在 AndroidManifest.xml 中添加<uses-permission android:name="android.permission.INTERNET" />来允许应用访问网络资源。

AndroidManifest.xml 中代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.AndroidBookProject2" android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/
    app name">
        <!--欢迎窗体-->
        <activity android:name=".Welcome" android:label="@string/
        app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!--推荐商品窗体-->
        <activity android:name=".ViewTuiJian" android:label="@string/
        app name">
        </activity>
        <!--家电办公商品窗体-->
        <activity android:name=".ViewJiaDian" android:label="@string/
        app_name">
        </activity>
        <!--手机数码商品窗体-->
        <activity android:name=".ViewShouJi" android:label="@string/
        app name">
        </activity>
        <!--电脑办公商品窗体-->
        <activity android:name=".ViewDianNao" android:label="@string/
        app_name">
        </activity>
        <!--商品详情窗体-->
        <activity android:name=".ShangPinDetailView" android:label=
        "@string/app name">
        </activity>
        <!--购物车列表窗体-->
        <activity android:name=".CartListView" android:label="@string/
        app_name">
        </activity>
        <!--订单详情窗体-->
        <activity android:name=".BillDetailView" android:label="@string/
```

```
        app_name">
        </activity>
        <!--登录窗体-->
        <activity android:name=".ViewLogin" android:label="@string/
        app_name">
        </activity>
        <!--订单列表窗体-->
        <activity android:name=".BillListView" android:label="@string/
        app_name">
        </activity>
        <!--主窗体-->
        <activity android:name=".ViewMain" android:label="@string/
        app_name">
        </activity>
    </application>
<!--添加允许访问网络资源权限-->
    <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

14.5　手机客户端的架构介绍

在手机客户端开发之前，首先对手机客户端的设计进行简单介绍，以帮助读者能很好地理解项目的开发流程及设计思想，希望能仔细阅读本节内容，在整体上理解本项目的设计。客户端的类设计按功能不同可以分为 3 部分，分别是客户端实体类、客户端工具类、客户端界面相关类。下面简要介绍各个类的功能。

14.5.1　客户端实体类简要介绍

客户端涉及对用户信息、商品信息、订单信息的处理，在设计上通过实体类来对用户、商品、订单进行描述，主要有以下 3 个实体类。

- 商品类 Goods：描述商品的基本信息，提供修改商品信息，获取商品信息的方法。
- 订单类 BillEntity：描述订单的基本信息，提供修改订单信息，获取订单信息的方法。
- 用户实体类 User：描述用户的基本信息，提供修改用户信息，获取用户信息的方法。

14.5.2　客户端工具类简要介绍

在项目实现过程中，涉及请求服务器资源，对中文乱码进行处理，购物车资源的存储。分别涉及下面 3 个工具类。

- ShopUtils 类：手机客户端提交订单信息时，如果有中文信息，提交到服务器上会是中文乱码，这里通过 ShopUtils 类对字符进行编码，在服务器端进行解码，实现解决中文乱码问题。
- DataShare 类：本实例中购物车效果的模拟是通过全局变量保存在手机内存中，在

该类中，定义了购物车列表的全局变量。

- ConnectWeb 类：该类负责从手机端发送请求到网站服务器端，请求服务器上的数据资源。

14.5.3　客户端界面相关类简要介绍

下面是手机端相关界面设计类的功能介绍。

- Welcome 类：项目启动后，欢迎窗体类。欢迎窗体显示 2 秒钟跳到主窗体 ViewMain 类。
- ViewMain 类：主窗体类，可以实现不同列表之间的切换。
- ViewTuiJian 类：展示推荐商品信息列表的窗体。该类通过工具类 ConnectWeb 类中提供的方法，请求服务器资源，返回列表信息，展示在窗体上。
- ViewJiaDian 类：展示家电商品信息列表的窗体。该类通过工具类 ConnectWeb 类中提供的方法，请求服务器资源，返回列表信息，展示在窗体上。
- ViewShouJi 类：展示手机数码商品信息列表的窗体。该类通过工具类 ConnectWeb 类中提供的方法，请求服务器资源，返回列表信息，展示在窗体上。
- ViewDianNao 类：展示电脑办公商品信息列表的窗体。该类通过工具类 ConnectWeb 类中提供的方法，请求服务器资源，返回列表信息，展示在窗体上。
- ShangPinDetailView 类：展示推荐商品信息列表的窗体。
- CartListView 类：展示购物车信息列表的窗体。该类读取 DataShare 类中购物车列表的共有变量，然后将列表信息展示在窗体中。
- ViewLogin 类：登录窗体。该类通过 ConnectWeb 类中提供的方法，向服务器端验证用户名和密码是否输入正确。
- BillDetailView 类：订单详情列表窗体。
- BillListView 类：订单列表窗体。该类通过 ConnectWeb 类中提供的方法，请求服务器端的订单信息列表，并显示在窗体上。

14.6　客户端实体类代码实现

上一节中简单介绍了项目中涉及的客户端实体类及功能，这里我们详细看一下客户端实体类的代码实现。

14.6.1　商品实体类设计及实现

Goods 类用来保存商品信息，需要注意的是，该类需要实现 Serializable 序列化接口，关于 Serializable 的介绍，大家可以参考 JavaSE 相关书籍来了解。

这里需要对该类的几个属性进行说明：

- 属性 bcount 表示商品的人气，即该商品总的购买次数。
- 属性 buyCount 表示当前用户购买了几件该商品。

- 属性 type 表示商品类型，分 3 类，即 1 表示家用电器，2 表示手机数码，3 表示电脑办公。

商品实体类 Goods，Java 代码如下：

```
package com.AndroidBookProject2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class Goods implements Serializable{
    private static final long serialVersionUID = 1L;
    private Integer id;                         //数据库编号
    private String brand;                       //名称
    private Float price;                        //价格
    private Float discount;                     //折扣
    private Integer bcount;                     //购买次数
    private String des;                         //描述
    private String pic;                         //图片名称
    private String dir;                         //图片路径
    private String gid;                         //商品编号
    private Integer type;                       //商品类型
    private Integer pop;                        //是否推荐
    private Integer buyCount;                   //当前用户对商品实际购买次数
……//该处省略了成员变量的 getXXX()和 setXXX()方法的代码，读者可自行查阅随书光盘
     中的源代码
}
```

14.6.2　订单实体类设计及实现

订单实体类定义了订单的基本信息，以及商品列表集合。这里需要对该类的几个属性进行说明。

- 商品列表属性 glist：保存该订单的商品信息，一个订单下可以有多个商品记录。
- 订单状态 state：默认是 waiting，即等待发货状态。

订单实体类 BillEntity，Java 代码如下：

```
package com.AndroidBookProject2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class BillEntity {
    private Integer id=0;                                   //数据库编号
    private String state="";                                //订单状态
    private String btime="";                                //送货时间
    private String btype="";                                //付款方式
    private String ctime="";                                //订单时间
    private List<GoodsListEntity> glist=new ArrayList<GoodsListEntity>();
                                                            //商品列表
……//该处省略了成员变量的 getXXX()和 setXXX()方法的代码，读者可自行查阅随书光盘
    中的源代码
}
```

14.6.3　用户实体类设计及实现

用户实体类设计较为简单，只有用户 id、用户昵称、用户密码 3 个属性。用户实体类 User 的代码如下：

```
package com.AndroidBookProject2;

public class User {
    private int id=0;                                          //用户 id
    private String uid="";                                     //用户昵称
    private String userPwd="";                                 //用户密码
……//该处省略了成员变量的 getXXX()和 setXXX()方法的代码，读者可自行查阅随书光盘
    中的源代码
}
```

14.7　编码转换类的设计及实现

当手机端提交中文信息到服务器端时，服务器端会出现乱码，解决这一问题需要在手机端对中文进行转码，在服务器端对数据解码才可以实现。手机端通过 Integer.toHexString()方法对中文字符串编码。ShopUtils 类的具体代码实现如下：

```
package com.AndroidBookProject2;
public class ShopUtils {
    /**
     * 转换编码
     */
    public static String changeToUnicode(String str) {
        StringBuffer strBuff = new StringBuffer();
        for (int i = 0; i < str.length(); i++) {
            String temp = Integer.toHexString(str.charAt(i));
            if (temp.length() != 4) {
                temp = "00" + temp;
            }
            if (temp.equals("00d")) {
                temp = "0" + temp;
            }
            if (temp.equals("00a")) {
                temp = "0" + temp;
            }
            strBuff.append(temp.substring(0, temp.length() - 2));
            strBuff.append(temp.substring(temp.length() - 2, temp.
            length()));
        }
        String returnData = strBuff.toString();
        return returnData;
    }
}
```

14.8　公共类的设计及实现

公共类 DataShare 中，提供了两个全局变量 user 和 shopList，分别用来存储用户信息及购物车列表信息。用户登录成功后，将用户信息保存到全局变量 user 中，在图 14.8 中，单击“结算”按钮结算时，通过判断 user 中是否有用户信息来决定程序的流程。

该类中提供了两个方法，isExistGoods()方法用来判断购物车上是否已经添加某一商品。getCartListMoney()方法计算购物车上所有商品的总价。

购物车列表及当前用户实体类 DataShare，Java 代码如下：

```
package com.AndroidBookProject2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class DataShare {
    public static User user=new User();           //保存当前登录用户的信息

    public static List<Goods> shopList=new ArrayList<Goods>();//购物车列表
    /**
     * 判断是否已经添加了一件该商品 返回值
     * @param id: 商品编号
     * @return: -1: 未添加过该商品，否则已添加过该商品
     */
    public static int isExistGoods(int id){
        for(int i=0;i<shopList.size();i++){
            if(shopList.get(i).getId()==id){
                return i;
            }
        }
        return -1;
    }
    /**
     * 获取购物车上商品的总价格
     * @return
     */
    public static float getCartListMoney(){
        float money=0.0f;
        for(int i=0;i<shopList.size();i++){

    money=money+shopList.get(i).getPrice()*shopList.get(i).getBuyCount()
;
        }
        return money;
    }
}
```

14.9　手机端请求服务器数据类的设计及实现

工具类 ConnectWeb 负责手机端请求服务器端数据资源。connWeb()方法请求服务器资源后返回结果集字符串，getPopList()方法将结果集字符串转换为 JSON 对象，解析返回商品列表信息。具体代码实现如下：

```
package com.AndroidBookProject2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ConnectWeb {
    private static String path = "http://192.168.1.8:8080/AndroidWeb/";
                                                        //服务器端程序地址

    //访问网站数据库获取数据
    private String connWeb(String url) {
        String str = "";
        try {
            HttpGet request = new HttpGet(url);     //创建 HttpGet 对象
            HttpClient httpClient = new DefaultHttpClient();
                                                        //创建 HttpClient 对象
```

```
            HttpResponse response = httpClient.execute(request);
                                                        //获取 HttpResponse 对象
            if (response.getStatusLine().getStatusCode() == 200) {
                                                        //判断服务器端返回状态
                str = EntityUtils.toString(response.getEntity());
                                                        //获得服务器端返回数据
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return str;
    }

    //获取推荐商品
    public List<Goods> getPopList() {
        List<Goods> mylist = new ArrayList<Goods>();    //定义 Goods 对象集合
        String url = path + "goodsAction.action?type=pop";
                                                        //准备请求地址参数
        String str = connWeb(url);                      //获取返回结果

        try {
            JSONObject job = new JSONObject(str);           //转换 JSONObject 对象
            JSONArray jay = job.getJSONArray("glist");     //获取 JSONArray 对象
            for (int i = 0; i < jay.length(); i += 1) {
                JSONObject temp = (JSONObject) jay.get(i);
                                                    //获取 JSONArray 中单个对象
                Goods goods = new Goods();          //定义 Goods 对象并赋值
                goods.setId(temp.getInt("id"));
                goods.setBrand(temp.getString("brand"));
                goods.setPrice((float) temp.getDouble("price"));
                goods.setDiscount((float) temp.getDouble("discount"));
                goods.setBcount(temp.getInt("bcount"));
                goods.setDes(temp.getString("des"));
                goods.setPic(temp.getString("pic"));
                goods.setDir(path + temp.getString("dir"));
                goods.setGid(temp.getString("gid"));
                goods.setType(temp.getInt("type"));
                goods.setPop(temp.getInt("pop"));
                mylist.add(goods);                  //添加 Goods 对象到集合
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        return mylist;
    }

    //获取推荐商品
    //1 家用电器，2 手机数码，3 电脑办公
    public List<Goods> getTypeList(int type) {
        List<Goods> mylist = new ArrayList<Goods>();    //定义 Goods 对象集合
        String url = path + "goodsAction.action?type=type&gtype=" + type;
                                                        //准备请求地址参数
        String str = connWeb(url);                      //获取返回结果

        try {
            JSONObject job = new JSONObject(str);           //转换 JSONObject 对象
            JSONArray jay = job.getJSONArray("glist");     //获取 JSONArray 对象
```

```
            for (int i = 0; i < jay.length(); i += 1) {
                JSONObject temp = (JSONObject) jay.get(i);
                                                //获取 JSONArray 中单个对象
                Goods goods = new Goods();      //定义 Goods 对象并赋值
                goods.setId(temp.getInt("id"));
                goods.setBrand(temp.getString("brand"));
                goods.setPrice((float) temp.getDouble("price"));
                goods.setDiscount((float) temp.getDouble("discount"));
                goods.setBcount(temp.getInt("bcount"));
                goods.setDes(temp.getString("des"));
                goods.setPic(temp.getString("pic"));
                goods.setDir(path + temp.getString("dir"));
                goods.setGid(temp.getString("gid"));
                goods.setType(temp.getInt("type"));
                goods.setPop(temp.getInt("pop"));
                mylist.add(goods);              //添加 Goods 对象到集合
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        return mylist;
    }

    //用户登录判断
    public boolean userLogin(String uid,String pwd) {
        boolean pan=true;                           //定义 boolean 值
        String url = path + "usersAction.action?uid="+uid+"&pwd="+pwd;
                                                    //准备请求地址参数
        String str = connWeb(url);                  //获取返回结果

        try {
            JSONObject job = new JSONObject(str);   //转换 JSONObject 对象
            pan=job.getBoolean("msg");
        } catch (Exception e) {
            e.printStackTrace();
        }
        return pan;
    }

    //获取用户订单
    public List<BillEntity> getBillList(String uid) {
        List<BillEntity> mylist = new ArrayList<BillEntity>();
                                                    //定义BillEntity对象集合
        String  url  =  path  +  "billAction.action?type=list&uid="+uid;
                                                    //准备请求地址参数
        String str = connWeb(url);                  //获取返回结果

        try {
            JSONObject job = new JSONObject(str);         //转换JSONObject对象
            JSONArray jay = job.getJSONArray("blist");   //获取 JSONArray 对象
            for (int i = 0; i < jay.length(); i += 1) {
                JSONObject temp = (JSONObject) jay.get(i);
                                                //获取 JSONArray 中单个对象
                BillEntity be = new BillEntity();
                                                //定义 BillEntity 对象并赋值
                be.setId(temp.getInt("id"));
                be.setState(temp.getString("state"));
                be.setBtime(temp.getString("btime"));
```

```
                be.setBtype(temp.getString("btype"));
                be.setCtime(temp.getString("ctime"));
                List<GoodsListEntity> glist=new ArrayList<Goods-
                ListEntity>();
                JSONArray gl = job.getJSONArray("glist");
                for (int j = 0; j < gl.length(); j += 1) {
                    GoodsListEntity ge=new GoodsListEntity();
                    JSONObject gtemp = (JSONObject) gl.get(j);
                    ge.setGname(gtemp.getString("gname"));
                    ge.setGnum(gtemp.getInt("gnum"));
                    glist.add(ge);
                }
                be.setGlist(glist);
                mylist.add(be);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        return mylist;
    }

    //增加用户订单
    /*
     * uid 用户登录 id
     * gids 商品数据库编号，多个商品之间用,分开   如 1,5
     * gnums 商品数量，多个数量之间用,分开   如 1,1  注意一个商品数据库编号对应一个
     商品数量
     * btime  送货时间   周一至周五/周末
     * btype  付款方式   现金/信用卡
     * address  地址
     *
     */
    public boolean addBill(String uid,String gids,String gnums,String
    btime,String btype,String address) {
        boolean pan=true;
        //准备请求地址参数
        String url = path + "billAction.action?type=add&uid="+uid+"&gids=
        "+gids+"&gnums="
                              +gnums+"&btime="+btime+"&btype="+btype+
                              "&address="+address;
        String str = connWeb(url);                    //获取返回结果

        try {
            JSONObject job = new JSONObject(str);   //转换 JSONObject 对象
            pan=job.getBoolean("msg");
        } catch (Exception e) {
            e.printStackTrace();
        }
        return pan;
    }
}
```

14.10　欢迎窗体类的设计及实现

欢迎窗体实现较为简单，主要通过 Thread 类实现 2 秒钟后跳转到主窗体，这个功能也

是很多应用都会实现的效果。

14.10.1　欢迎窗体的框架设计

首先介绍欢迎窗体的代码设计框架，有利于大家更好地了解代码流程，具体代码如下：

```
package com.AndroidBookProject2
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
/**
 * 第一个欢迎窗体
 * */

public class Welcome extends Activity{

    /**
     * 重写 Activity 中的 onCreate 的方法。
     * 该方法是在 Activity 创建时被系统调用，是一个 Activity 生命周期的开始
     * @param savedInstanceState：保存 Activity 的状态的
     *      Bundle 类型的数据与 Map 类型的数据相似，都是以 key-value 的形式存储数据的
     * @return
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        ……//此处省略了初始化窗体事件，将在之后进行介绍
    }

    /**
     * 欢迎界面,2 秒钟后切换
     * @param
     * @return
     */
    public void welcome() {
        new Thread(new Runnable() {                     //创建线程
            public void run() {          //实现 Runnable 的 run()方法，即线程体
                try {
                    Thread.sleep(2000);             //欢迎界面暂停 2 秒钟
                    Message m = new Message();      //创建 Message 对象
                    logHandler.sendMessage(m);      //将消息放到消息队列中
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }).start();                                     //启动线程
    }

    //执行接收到的消息，执行的顺序是按照队列进行，即先进先出
    Handler logHandler = new Handler() {
        public void handleMessage(Message msg) {
            welcome1();                                 //显示 Logo 界面
        }
    };
    /**
     * 显示 Logo 界面
```

```
     * @param
     * @return
     */
    public void welcome1() {
        Intent it=new Intent();                           //实例化 Intent
        it.setClass(Welcome.this, ViewMain.class); //设置 Class
    startActivity(it);                                    //启动 Activity
    Welcome.this.finish();                                //结束 Welcome Activity
    }

    /**
     * 键盘按键按下是触发该方法
     * @param keyCode：被按下的键值即键盘码
     *        event：按键事件的对象
     * @return
     */
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if(keyCode==4 ){                                  //按下“返回”按键
            android.os.Process.killProcess(android.os.Process.myPid());
                                                          //让程序完全退出应用
        }
        return super.onKeyDown(keyCode, event);
    }
}
```

14.10.2　欢迎窗体的初始化工作

欢迎窗体是以全屏的方式展现，需要在 Activity 的 onCreate()方法中隐藏标题栏和状态栏。窗体初始化代码如下：

```
protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        //返回当前 Activity 的 Window 对象，Window 类中概括了 Android 窗口的基本属性
          和基本功能
        final Window win = getWindow();
        //隐藏状态栏
    win.setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,WindowManager.
    LayoutParams.FLAG_FULLSCREEN);
        requestWindowFeature(Window.FEATURE_NO_TITLE);      //隐藏标题栏
        this.setContentView(R.layout.welcome);              //设置布局资源
        //获取 welcome.xml 中 id 为 wpic 的 ImageView 组件
        ImageView iv=(ImageView)this.findViewById(R.id.wpic);
        iv.setImageResource(R.drawable.welcome);
                                              //设置 ImageView 上显示的资源
        welcome();//欢迎界面
    }
```

14.11　应用主窗体类的设计及实现

主窗体类 ViewMain，项目中的所有功能通过该窗体进行切换展示，该类继承了 TabActivity。通过 tabHost.setCurrentTab(0);指定默认显示第一个 tab 页，即推荐商品列表窗

体。具体实现代码如下：

```
package com.AndroidBookProject2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ViewMain extends TabActivity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        final TabHost tabHost = getTabHost();
                                       //获取当前 Activity 的 TabHost 对象
    //添加“推荐商品”标签页，设置标签页上的图片为 R.drawable.popular，点击该标签页
     跳到 ViewTuiJian 窗体
       tabHost.addTab(tabHost.newTabSpec("tab1")
              .setIndicator("推荐商品", getResources().getDrawable
              (R.drawable.popular))
              .setContent(new Intent(this, ViewTuiJian.class)));
    //添加“家用电器”标签页，设置标签页上的图片为 R.drawable. home，点击该标签页跳
      到 ViewJiaDian 窗体
       tabHost.addTab(tabHost.newTabSpec("tab2")
              .setIndicator("家用电器", getResources().getDrawable(R.
              drawable.home))
              .setContent(new Intent(this, ViewJiaDian.class)));
    //添加“手机数码”标签页，设置标签页上的图片为 R.drawable.mobile，点击该标签页
      跳到 ViewShouJi 窗体
       tabHost.addTab(tabHost.newTabSpec("tab3")
              .setIndicator("手机数码", getResources().getDrawable(R.
              drawable.mobile))
              .setContent(new Intent(this, ViewShouJi.class)));
    //添加“电脑办公”标签页，设置标签页上的图片为 R.drawable.computer，点击该标签
      页跳到 ViewDianNao 窗体
       tabHost.addTab(tabHost.newTabSpec("tab4")
              .setIndicator("电脑办公", getResources().getDrawable(R.
              drawable.computer))
              .setContent(new Intent(this, ViewDianNao.class)));
       tabHost.setCurrentTab(0);  //设置默认显示页为第一个标签页
    }
}
```

运行效果如图 14.13 所示。



图 14.13　主窗体

14.12　推荐商品列表窗体类的设计及实现

推荐商品列表窗体类 ViewTuiJian 的实现涉及以下几个技术点：

- 动态创建用来显示商品列表的组件 ListView。
- 通过 ListView 的 setOnItemClickListener 事件监听单击列表项事件。
- 将从服务器请求回来的数据显示在 ListView 中，通过自定义 showGoodsList()方法实现。
- 布局文件分为两部分，一部分是推荐商品列表展示的布局文件 res/layout/viewtuijian.xml；另一部分是推荐商品列表项布局文件 res/layout/tuijianrow.xml。当单击选择某一个列表项或者列表项获取某一个焦点时，列表项会改变背景，此效果通过 res/layout/trippoilistviewbg.xml 实现。

14.12.1　推荐商品列表的设计

推荐商品列表的设计思路及实现流程如下：

（1）首先通过 ConnectWeb 类中提供的 getPopList()方法，获取商品信息列表，并将其保存到商品集合 goodsList 变量中。

（2）然后创建 SimpleAdapter 适配器，数据源是 goodsList 集合中的数据信息。

（3）最后创建 ListView 组件，为该组件添加 SimpleAdapter 适配器，以便显示数据。

这里我们看一下推荐商品列表的详细代码实现：

```
package com.AndroidBookProject2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ViewTuiJian extends Activity {
    private LinearLayout myListLayout;
                                //声明用来显示商品列表 LinearLayout 布局变量
    private ListView tripListView; //声明用来显示商品列表的 ListView 变量
    private ProgressDialog myDialog; //声明 ProgressDialog 类型变量
    private List<Goods> goodsList; //声明 List 类型变量，用来保存商品列表集合

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.viewtuijian);
                                //加载 viewtuijian.xml 资源文件
        myListLayout = (LinearLayout) this.findViewById(R.id.tripList);
                                //获取资源文件中的 LinearLayout

        tripListView = new ListView(this); //创建 ListView 对象
        //创建 LinearLayout.LayoutParams 类型对象
        LinearLayout.LayoutParams tripListViewParam = new LinearLayout.
        LayoutParams(
                LinearLayout.LayoutParams.FILL_PARENT,
                LinearLayout.LayoutParams.FILL_PARENT);
        tripListView.setCacheColorHint(Color.WHITE);
```

```
        myListLayout.addView(tripListView, tripListViewParam);
                                //将 tripListView 添加到 myListLayout 布局上
        getGoodsList();         //读取商品列表

        tripListView.setOnItemClickListener(new OnItemClickListener() {
                                //tripListView 列表项单击事件

                    @Override
                    public void onItemClick(AdapterView<?> arg0, View arg1,
                            int position, long id) {
                        //TODO Auto-generated method stub
                        Goods theGood = goodsList.get(position);
                                //获取当前列表项选中的商品
                        Intent it = new Intent();       //创建 Intent 对象
                        Bundle bundle = new Bundle();   //创建 Bundle 对象
                        it.setClass(ViewTuiJian.this, ShangPinDetailView.
                        class);
                        bundle.putSerializable("GoodObj",
                                (Serializable) theGood);
                        it.putExtras(bundle);
                        startActivity(it);              //启动窗体
                    }
                });
    }

    /**
     * 读取商品列表数据
     */
    private void getGoodsList() {
        myDialog = ProgressDialog.show(ViewTuiJian.this, "请稍等...", "数
        据检索中...",
                true);
        new Thread() {
            public void run() {
                try {
                    goodsList = new ConnectWeb().getPopList();
                                                    //获取推荐商品列表
                    Message m = new Message();
                    listHandler.sendMessage(m);
                } catch (Exception e) {
                    e.printStackTrace();
                } finally {
                    myDialog.dismiss();
                }
            }
        }.start();
    }

    Handler listHandler = new Handler() {
        public void handleMessage(Message msg) {
            if (goodsList.size() == 0) {
                return;
            }
            showGoodsList();//该方法的实现，将在之后进行介绍
        }
    };
}
```

14.12.2　推荐商品列表 ListView 数据填充

推荐商品列表展示在 ListView 组件上，推荐商品列表数据保存在 goodsList 集合中，通过 getTripList()方法，将商品名称、商品价格、商品折扣以(key,value)的形式保存到集合中，作为 SimpleAdapter 的数据源。具体实现代码如下：

```
/**
* 填充商品列表适配器
*/
    public void showGoodsList() {
        SimpleAdapter adapter = new SimpleAdapter(this, getTripList(),
                R.layout.tuijianrow, new String[] { "img", "name", "money",
                        "zhe" }, new int[] { R.id.tripImg, R.id.tripTitle,
                        R.id.tripSegName, R.id.tripProv });
        tripListView.setAdapter(adapter);
                                        //为 tripListView 添加适配器 adapter
        adapter.setViewBinder(new ViewBinder() {
            public boolean setViewValue(View arg0, Object arg1,
                    String textRepresentation) {
                //判断 arg0 是否为 ImageView 类型，arg1 是否为 Bitmap 类型
                if ((arg0 instanceof ImageView) & (arg1 instanceof Bitmap))
{
                    ImageView imageView = (ImageView) arg0;
                    Bitmap bitmap = (Bitmap) arg1;
                    imageView.setImageBitmap(bitmap);
                                        //设置 imageView 上显示的图片
                    return true;
                } else {
                    return false;
                }
            }
        });
    }
    //获取商品列表的 List，List 数据格式为 Map 类型
    public List<Map<String, Object>> getTripList() {
        List<Map<String, Object>> list = new ArrayList<Map<String,
        Object>>();
        //遍历商品列表集合，以(key,value)形式保存到 Map 集合中
        for (int i = 0; i < goodsList.size(); i += 1) {
            Map<String, Object> map = new HashMap<String, Object>();
            Goods goods = goodsList.get(i);     //获取集合中的元素

            try {
                URL picUrl = new URL(goods.getDir() + "/" + goods.getPic());
                Bitmap pngBM = BitmapFactory.decodeStream(picUrl.
                openStream());
                map.put("img", pngBM);          //将商品图片添加到 map 中
            } catch (Exception e) {
                e.printStackTrace();
            }
            map.put("name", "商品名称："+goods.getBrand());
                                                //将商品名称添加到 map 中
            map.put("money", "商品价格："+"￥" + goods.getPrice());
                                                //将商品价格添加到 map 中
            map.put("zhe", "商品折扣："+goods.getDiscount());
```

```
                                              //将商品折扣添加到 map 中
            list.add(map);
        }
        return list;
    }
```

家用电器商品列表窗体类 ViewJiaDian、手机数码商品列表窗体类 ViewShouJi、电脑办公商品列表窗体类 ViewDianNao 代码上的实现和推荐商品列表窗体类 ViewTuiJian 的实现相似，都是请求服务器资源，展示在窗体列表上。具体的代码，请读者自行查阅随书光盘中的源代码，这里不再赘述。

14.13　商品详情信息窗体类的设计及实现

商品详情信息类 ShangPinDetailView，该窗体展示的商品详情信息从 Bundle 中获取，另外可以通过窗体的 menu 查看购物车列表信息。

14.13.1　商品详情信息窗体类的框架设计

商品详情信息窗体的设计流程具体如下：

（1）获取商品信息的 Bundle 对象，从 Bundle 中获取产品对象 Goods 信息，显示到窗体相关组件上。

（2）添加购物车菜单，并对菜单事件进行处理。

（3）实现添加到购物车功能。

具体实现代码如下：

```
package com.AndroidBookProject2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ShangPinDetailView extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.shangpindetailview); //加载布局资源文件
        getGoodDetail();                                   //获取商品详细信息
    }

    /**
     * 获取商品详细信息
     */
    public void getGoodDetail() {
        Bundle bundle = this.getIntent().getExtras();  //获取 Bundle
        final Goods theGood = (Goods) bundle.getSerializable("GoodObj");
                                              //获取 Bundle 中的商品对象
        //显示详情信息
        TextView goodsName = (TextView) this.findViewById(R.id.goodsName);
                                              //获取资源文件中的 TextView
        goodsName.setText(theGood.getBrand()); //将商品名称显示在 TextView
        //商品图片
```

```
        ImageView goodsPic = (ImageView) this.findViewById(R.id.goodsPic);
        try {
            URL picUrl = new URL(theGood.getDir() + "/" + theGood.getPic());
            Bitmap pngBM = BitmapFactory.decodeStream(picUrl.
            openStream());
            goodsPic.setImageBitmap(pngBM);
        } catch (Exception e) {
            e.printStackTrace();
        }
        TextView goodsNum = (TextView) this.findViewById(R.id.goodsNum);
                                                            //商品编号
        goodsNum.setText("商品编号:" + theGood.getGid());
        TextView goodsPrice = (TextView) this.findViewById(R.id.
        goodsPrice);                                        //价格
        goodsPrice.setText("价      格:" + "￥" + theGood.getPrice().
        toString());
        TextView goodsDcount = (TextView) this.findViewById(R.id.
        goodsDcount);                                       //人气
        goodsDcount.setText("人      气:" + theGood.getBcount().
        toString());
        TextView goodsDiscount = (TextView) this
                .findViewById(R.id.goodsDiscount);          //折扣
        goodsDiscount.setText("折      扣:" + theGood.getDiscount().
        toString());
        TextView descTip = (TextView) this.findViewById(R.id.descTip);
                                                            //详情
        TextPaint textPaint = descTip.getPaint();
        textPaint.setFakeBoldText(true);
        TextView goodsDes = (TextView) this.findViewById(R.id.goodsDes);
        goodsDes.setText(theGood.getDes());
        Button addToCard = (Button) this.findViewById(R.id.addToCard);
                                                            //放入购物车按钮
        addToCard.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                ……//此处省略了添加购物车功能代码，将在之后进行介绍
            }
        });
    }
……//此处省略了窗体菜单代码，将在之后进行介绍
}
```

14.13.2　添加购物车功能的实现

在本项目中，购物车数据存储在内存中，由 DataShare 类中的 shopList 集合对象保存。在添加购物车的时候，需要注意的问题是要避免重复添加商品，如果 1 号商品已经添加到购物车，用户再次添加的时候，只需要将 1 号商品的购买次数增加 1。购物车功能具体实现代码如下：

```
//放入购物车按钮
        addToCard.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(ShangPinDetailView.this, "放入购物车成功",
                        Toast.LENGTH_LONG).show();
                //TODO Auto-generated method stub
```

```
                int index = DataShare.isExistGoods(theGood.getId());
                if (index != -1) {                        //已添加过该商品
                    //商品购买数量+1
                    DataShare.shopList.get(index).setBuyCount(
                            DataShare.shopList.get(index).getBuyCount() +1);
                } else {                                  //未添加过该商品
                    theGood.setBuyCount(1);
                    DataShare.shopList.add(theGood);
                }
            }
        });
```

14.13.3　菜单设计与实现

该窗体中提供了“购物车列表”菜单，用户可以查看目前购物车上商品的信息。具体菜单设计与实现，代码如下：

```
//菜单
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuItem mnuxq = menu.add(0, 0, 0, "购物车");
        mnuxq.setIcon(R.drawable.cart);
        return super.onCreateOptionsMenu(menu);
    }
    //菜单响应事件
    public boolean onOptionsItemSelected(MenuItem item) {
        super.onOptionsItemSelected(item);
        switch (item.getItemId()) {
        case 0:
            Intent it = new Intent();
            it.setClass(ShangPinDetailView.this, CartListView.class);
            startActivity(it);
            break;
        }
        return true;
    }
```

商品详情信息窗体，运行效果如图 14.14 所示。



图 14.14　商品详情展示窗体

14.14　购物车列表窗体类的设计及实现

单击图 14.6 中的“放入购物车”按钮，会将当前商品添加到购物车上。单击“menu”按钮，会在窗体上出现“购物车”菜单，如图 14.7 所示。购物车列表信息保存在全局变量 shopList 集合中，通过读取 shopList 信息，实现展示购物车中的每件商品。

14.14.1　购物车列表窗体的框架设计

购物车列表窗体的实现大概包括以下几方面：

- 动态创建 ListView。
- 获取购物车列表信息作为 SimpleAdapter 的数据源。
- 为 ListView 添加 SimpleAdapter。
- 结算金额之前要判断用户状态，必须是登录状态才可以进行结算。

具体代码如下：

```
package com.AndroidBookProject2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class CartListView extends Activity {
    private LinearLayout myListLayout;
                                //声明用来显示购物车列表的 LinearLayout 类型变量
    private ListView tripListView; //声明用来显示购物车列表的 ListView 类型变量

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.viewcartlist);    //加载布局资源文件
        myListLayout = (LinearLayout) this.findViewById(R.id.cartList);
                                       //获取资源文件中 LinearLayout
        tripListView = new ListView(this);             //创建 ListView
        //创建布局参数
        LinearLayout.LayoutParams tripListViewParam = new LinearLayout.
        LayoutParams(
                LinearLayout.LayoutParams.FILL_PARENT,
                LinearLayout.LayoutParams.FILL_PARENT);
        tripListView.setCacheColorHint(Color.WHITE);
        myListLayout.addView(tripListView, tripListViewParam);
                                       //将 ListView 添加到 LinearLayout 上
        showGoodsList();               //获取商品列表

        Button accountsBut = (Button) this.findViewById(R.id.accountsBut);
                                       //加载资源文件中的 Button
        accountsBut.setOnClickListener(new OnClickListener(){
                                       //Button 的单击事件

            @Override
            public void onClick(View v) {
```

```
                ……//此处省略了结算功能代码，将在之后进行介绍
            }
        });
    }
    //填充商品列表适配器
    public void showGoodsList() {
        SimpleAdapter adapter = new SimpleAdapter(this, getTripList(),
                R.layout.cartlistrow, new String[] { "img", "num", "name",
                        "price" }, new int[] { R.id.cartImg, R.id.cartNum,
                        R.id.cartName, R.id.cartPrice });
        tripListView.setAdapter(adapter);  //为 ListView 添加 Adatpter
//setViewBinder()方法设置 binder 用于绑定数据到视图，参数为用于绑定数据到视图的
binder
        adapter.setViewBinder(new ViewBinder() {
            public boolean setViewValue(View arg0, Object arg1,
                    String textRepresentation) {
                if ((arg0 instanceof ImageView) & (arg1 instanceof Bitmap))
{
                    ImageView imageView = (ImageView) arg0;
                    Bitmap bitmap = (Bitmap) arg1;
                    imageView.setImageBitmap(bitmap);
                                            //设置 imageView 组件显示的图片
                    return true;
                } else {
                    return false;
                }
            }
        });
    }
    //获取商品列表集合
    public List<Map<String, Object>> getTripList() {
        List<Map<String, Object>> list = new ArrayList<Map<String,
        Object>>();
        //遍历购物车列表
        for (int i = 0; i < DataShare.shopList.size(); i += 1) {
            Map<String, Object> map = new HashMap<String, Object>();
            Goods goods = DataShare.shopList.get(i);
                                        //获取购物车列表指定位置元素
            try {
                URL picUrl = new URL(goods.getDir() + "/" + goods.getPic());
                                        //加载服务器图片
                Bitmap pngBM = BitmapFactory.decodeStream(picUrl.
                openStream());
                map.put("img", pngBM);
            } catch (Exception e) {
                e.printStackTrace();
            }
            map.put("num", "商品编号：" + goods.getGid());
            map.put("name", "商品名称：" + goods.getBrand());
            map.put("price", "￥" + goods.getPrice()+"  数量："+goods.get-
            BuyCount());
            list.add(map);
        }
        return list;
    }
}
```

14.14.2　结算功能实现

用户单击“结算”按钮后，首先需要判断用户全局变量 user 对象的用户昵称 uid 是否存在。如果有相关的用户昵称，就认为用户已经登录了应用，则进入提交订单窗体，否则认为用户没有登录，则跳到登录窗体。具体代码如下：

```
accountsBut.setOnClickListener(new OnClickListener() {//Button 的单击事件
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                //判断用户是否登录
                if (DataShare.user.getUid().equals("")) {
                    Intent intent = new Intent();
                    intent.setClass(CartListView.this, ViewLogin.class);
                    startActivity(intent);
                    Toast.makeText(CartListView.this, "请先登录", Toast.
                    LENGTH_LONG).show();
                } else {
                    Intent intent = new Intent();
                    intent.setClass(CartListView.this, BillDetailView.
                    class);
                    startActivity(intent);
                }
            }
```

购物车列表窗体，运行效果如图 14.15 所示。



图 14.15　购物车列表窗体

14.15　登录窗体类的设计及实现

单击图 14.8 中的“结算”按钮，如果在手机客户端登录了该应用，则会显示提交订单窗体，否则显示登录窗体。用户信息保存在全局变量 user 中，通过判断 user 的值可以知道

当前用户是否为登录状态。

14.15.1　登录窗体的框架设计

登录窗体代码流程具体如下：

```
package com.AndroidBookProject2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class ViewLogin extends Activity {
    private EditText username;          //声明用户名文本框 EditText 类型变量
    private EditText password;          //声明密码文本框 EditText 类型变量
    private Button loginBut;            //声明登录按钮 Button 类型变量
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.viewlogin);
                                        //加载布局资源文件 viewlogin.xml

        username=(EditText) this.findViewById(R.id.username);
                                        //加载布局资源文件中的 EditText 组件
        password=(EditText) this.findViewById(R.id.password);
                                        //加载布局资源文件中的 EditText 组件
        loginBut=(Button) this.findViewById(R.id.loginBut);
                                        //加载布局资源文件中的 Button 组件
        loginBut=(Button) this.findViewById(R.id.loginBut);
                                        //加载布局资源文件中的 Button 组件
        loginBut.setOnClickListener(new OnClickListener(){
                                        //Button 组件的单击事件
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                ……//此处省略了登录功能实现，将在之后进行介绍
            }

        });

    }

}
```

14.15.2　登录功能代码实现

单击“登录”按钮时，需要去服务器端验证用户名和密码是否正确，通过 ConnectWeb 类的 userLogin(uid,password)方法，将客户输入的用户名和密码以参数的方式传递给服务器。如果用户名和密码无误，则进入订单列表窗体。具体代码实现如下：

```
loginBut.setOnClickListener(new OnClickListener(){ //Button 组件的单击事件
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                //验证用户名和密码是否正确
                User theuser=new ConnectWeb().userLogin(username.
                getText().toString(),password.getText().toString());
```

```
                if(theuser!=null){
                    DataShare.user.setId(theuser.getId()); //设置用户 id
                    DataShare.user.setUid(theuser.getUid());//设置用户昵称
                    DataShare.user.setUserPwd(theuser.getUserPwd());
                                                        //设置用户密码
                    Toast.makeText(ViewLogin.this, "登录成功", Toast.
                    LENGTH_LONG).show();
                    //登录成功后进入购物车
                    Intent intent = new Intent();
                    intent.setClass(ViewLogin.this, CartListView.class);
                    startActivity(intent);                  //启动窗体
                }else{
                    Toast.makeText(ViewLogin.this, "用户名或者密码错误",
                    Toast.LENGTH_LONG).show();
                }
            }

        });
```

14.16　提交订单窗体类的设计及实现

如果用户为登录状态，在购物车列表中单击“结算”按钮，会到提交订单窗体。用户在窗体上填写的数据信息，通过 ConnectWeb 类中的 addBill()方法添加到服务器的数据库中。

14.16.1　提交订单窗体类的框架设计

提交订单窗体的实现具体流程如下：

（1）DataShare 类中提供了 getCartListMoney()方法，可以获取当前购物车上的商品金额。

（2）提交订单时，通过 ConnectWeb 类中的 addBill()方法，将订单信息保存到服务器端。

（3）提交订单窗体提供了菜单，可以显示订单列表。

具体代码实现如下：

```
package com.AndroidBookProject2;
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class BillDetailView extends Activity{
    private Spinner payMoneyWaySpinner; //声明付款方式的 Spinner 类型变量
    private Spinner sendTimeSpinner;    //声明送货时间的 Spinner 类型变量
    private TextView payMoneyTextView;  //声明用来显示应付金额的TextView类型变量
    private EditText personAddr;        //声明地址信息的 EditText 类型变量
    private Button submitBill;          //声明提交按钮的 Button 类型变量
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.viewbilldetail);
                                //加载布局资源文件 viewbilldetail.xml
```

```
        //付款方式的 Spinner
        //获取资源文件中的 Spinner 组件
        payMoneyWaySpinner=(Spinner) this.findViewById(R.id.payMoneyWay);
        //创建 ArrayAdapter 适配器对象，数据来源于数组 pay_MoneyWayArr
        ArrayAdapter<CharSequence> payMoneyadapter = ArrayAdapter.
        createFromResource(
                this, R.array.pay MoneyWayArr, android.R.layout.
                simple spinner item);
        payMoneyadapter.setDropDownViewResource(android.R.layout.
        simple spinner dropdown item);
        payMoneyWaySpinner.setAdapter(payMoneyadapter);
                                        //为 payMoneyWaySpinner 添加适配器

        //送货时间的 Spinner
        sendTimeSpinner=(Spinner) this.findViewById(R.id.sendTime);
                                        //获取资源文件中的 Spinner 组件
        //创建 ArrayAdapter 适配器对象，数据来源于数组 pay_MoneyWayArr
        ArrayAdapter<CharSequence> sendTimeadapter = ArrayAdapter.
        createFromResource(
                this, R.array.send TimeArr, android.R.layout.
                simple spinner item);
        sendTimeadapter.setDropDownViewResource(android.R.layout.
        simple spinner dropdown item);
        sendTimeSpinner.setAdapter(sendTimeadapter);
                                        //为 sendTimeSpinner 添加适配器

        //获取应付金额 TextView
        payMoneyTextView=(TextView) this.findViewById(R.id.payMoney);
                                        //获取资源文件中的 TextView 组件
    payMoneyTextView.setText(DataShare.getCartListMoney()+"元");
                                        //设置 payMoneyTextView 组件显示文字
        //获取地址信息
        personAddr=(EditText) this.findViewById(R.id.personAddr);
                                        //获取资源文件中的 EditText 组件
        //提交订单
        submitBill=(Button) this.findViewById(R.id.submitBill);
                                        //获取资源文件中的 Button 组件
        submitBill.setOnClickListener(new OnClickListener(){
                                        //监听 Button 的单击事件

            @Override
            public void onClick(View v) {
                ……//此处省略了提交订单功能代码，将在之后进行介绍
            }
        });
    }
    //菜单
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuItem mnudt = menu.add(0, 0, 0, "订单中心");
        mnudt.setIcon(R.drawable.bill);
        return super.onCreateOptionsMenu(menu);
    }
    //菜单响应事件
    public boolean onOptionsItemSelected(MenuItem item) {

        super.onOptionsItemSelected(item);
```

```
        switch (item.getItemId()) {
        case 0:
            Intent it1 = new Intent();
            it1.setClass(BillDetailView.this, BillListView.class);
            startActivity(it1);
            break;
        }
        return true;
    }
}
```

14.16.2　提交订单功能实现

自定义方法 addBill()用来提交订单到服务器端，该方法需要传递 6 个参数。方法原型如下：

```
public boolean addBill(String uid,String gids,String gnums,String
btime,String btype,String address)
```

参数含义分别如下。

- uid：用户登录 id。
- gids：商品数据库编号，多个商品之间用逗号分开，如 1,5。
- gnums：商品数量，多个数量之间用逗号分开，如 1,1。注意一个商品数据库编号对应一个商品数量。
- btime：送货时间。
- btype：付款方式。
- address：地址。

首先遍历 shopList 集合组装 gids 和 gnums 字符串，然后调用 addBill()方法添加订单，代码如下：

```
submitBill.setOnClickListener(new OnClickListener() //监听 Button 的单击事件
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                //gids 保存订单编号字符串，订单编号字符串格式为编号 1，编号 2
                //gnums 保存购买次数字符串，购买次数字符串格式为次数 1，次数 2
                String gids="",gnums="";
                for(int i=0;i<DataShare.shopList.size();i++){
                    gids=gids+DataShare.shopList.get(i).getId()+",";
                    gnums=gnums+DataShare.shopList.get(i).getBuyCount()+",";
                }
                if(gids.length()>0){
                    gids=gids.substring(0, gids.length()-1);
                }
                if(gnums.length()>0){
                    gnums=gnums.substring(0, gnums.length()-1);
                }
                //添加订单
                new ConnectWeb().addBill(DataShare.user.getUid(), gids,
                gnums, ShopUtils.changeToUnicode(sendTimeSpinner.
                getSelectedItem().toString()), ShopUtils.changeToUnicode
                (payMoneyWaySpinner.getSelectedItem().toString()),
                ShopUtils.changeToUnicode(personAddr.getText().
```

```
                    toString()) );
                    Toast.makeText(BillDetailView.this, "订单提交完毕", Toast.
                    LENGTH LONG).show();
                }
            });
```

添加订单窗体，运行效果如图 14.16 所示。



图 14.16　提交订单信息

14.17　订单列表窗体类的设计及实现

在提交订单窗体，单击“menu”按钮，会出现订单列表菜单，单击该菜单会出现订单列表信息。订单列表是通过 ConnectWeb 类中的 getBillList()方法从服务器上请求订单数据，然后展示在 ListView 上。

14.17.1　订单列表窗体类框架设计

订单信息列表窗体的代码流程如下：

（1）创建 ListView 对象。

（2）从服务器端读取订单列表信息，通过 getBillList()方法实现。

（3）创建 SimpleAdapter 对象，数据源通过 getTripList()方法获取订单列表信息的(key,value)map 集合。

（4）为 ListView 添加 SimpleAdapter 适配器。

具体实现代码如下：

```
package com.AndroidBookProject2;
```

```
……//该处省略了部分类的导入代码，读者可自行查阅随书光盘中的源代码
public class BillListView extends Activity {
    private LinearLayout myListLayout;
                                //声明用来显示订单列表的LinearLayout类型变量
    private ListView tripListView; //声明用来显示订单列表的ListView类型变量
    private List<BillEntity> billList = new ArrayList<BillEntity>();
                                //声明List类型变量，保存订单集合
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.viewbilllist);    //加载布局资源文件
        //加载布局资源文件中的LinearLayout组件
        myListLayout = (LinearLayout) this.findViewById(R.id.billList);
        tripListView = new ListView(this);             //创建ListView
        //创建布局参数
        LinearLayout.LayoutParams tripListViewParam = new LinearLayout.
        LayoutParams(
                LinearLayout.LayoutParams.FILL_PARENT,
                LinearLayout.LayoutParams.FILL_PARENT);
        tripListView.setCacheColorHint(Color.WHITE);
        myListLayout.addView(tripListView, tripListViewParam);
                                    //将ListView添加到LinearLayout上
        getBillList();              //获取订单列表
    }
    ……//这里省略了getBillList()方法，读取订单列表功能代码，将在之后进行介绍
}
```

14.17.2　读取订单列表功能实现

订单列表的读取，通过 ConnectWeb 类中自定义 getBillList()方法实现，参数是 uid，订单列表读取完毕之后，调用 showBillList()创建 SimpleAdapter，并将 SimpleAdapter 添加到 ListView 中。具体代码如下：

```
//读取订单列表数据
    private void getBillList() {
        final ProgressDialog myDialog = ProgressDialog.show(BillListView.
        this,
                "请稍等...", "数据检索中...", true);
        new Thread() {
            public void run() {
                try {
                    //从服务器端读取用户订单列表
                    billList = new ConnectWeb().getBillList(DataShare.user
                            .getUid());
                    Message m = new Message();
                    listHandler.sendMessage(m);
                } catch (Exception e) {
                    e.printStackTrace();
                } finally {
                    myDialog.dismiss();                //关闭对话框
                }
            }
        }.start();
    }
    Handler listHandler = new Handler() {
```

```
        public void handleMessage(Message msg) {
            if (billList.size() == 0) {
                return;
            }
            showBillList();                          //填充订单列表适配器
        }
    };
    //填充订单列表适配器
    public void showBillList() {
        //声明 SimpleAdapter 适配器
        SimpleAdapter adapter = new SimpleAdapter(this, getTripList(),
                R.layout.billlistrow, new String[] { "billNum",
                "billState","billTime" }, new int[] { R.id.billNum,
                R.id.billState,R.id.billTime });
        tripListView.setAdapter(adapter);       //为 tripListView 添加适配器
    }

    public List<Map<String, Object>> getTripList() {

        List<Map<String, Object>> list = new ArrayList<Map<String,
        Object>>();
        for (int i = 0; i < billList.size(); i += 1) { //遍历订单列表集合
            Map<String, Object> map = new HashMap<String, Object>();
            BillEntity theBill = billList.get(i);
            String stateStr = "";
            if (theBill.getState().equals("waiting")) {
                                        //数据库中保存的 waiting 表示该订单为处理中
                stateStr = "处理中";
            }
            map.put("billNum", "订单编号: " + theBill.getId());
            map.put("billTime", "下单时间: " + theBill.getCtime());
            map.put("billState", "订单状态: " + stateStr);
            list.add(map);
        }
        return list;
    }
```

运行效果如图 14.17 所示。



图 14.17　订单列表窗体

14.18　项目技术难点及改进

到本节为止，本项目基本功能已经介绍完毕，项目在实现过程中的技术难点大概有以下几方面。

- 手机端请求服务器数据进行解析成JSON对象。这里需要读者掌握Android的HTTP技术，并要掌握 Android 中解析 JSON 数据技术。
- 手机端向服务器端保存中文数据，注意要对数据进行编码转换，在服务器端对数据再进行解析。否则中文数据无法正确地保存到服务器端的数据库中。

目前登录的用户信息的存储及购物车信息的存储保存在全局变量中，这样退出应用后这些资源会被释放。如果想退出应用，下次登录时，购物车上依然保留之前购选商品的信息，就需要将数据保存到手机数据库中。